

(19) World Intellectual Property Organization  
International Bureau



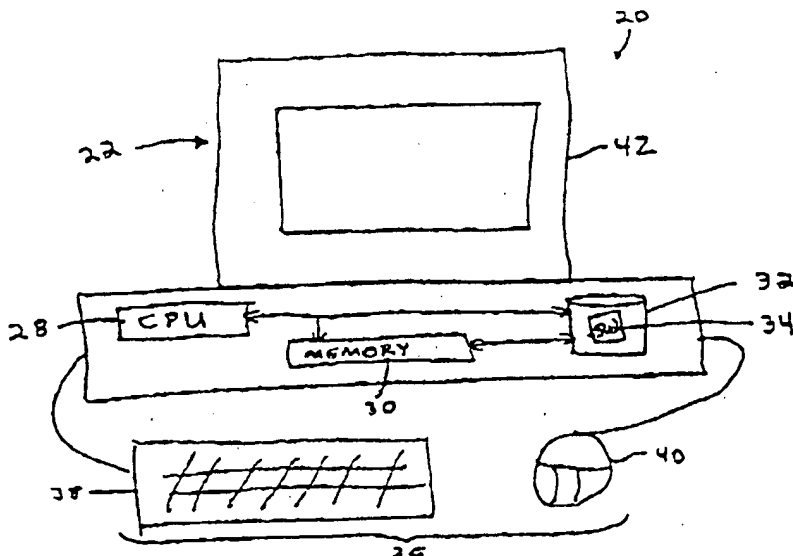
(43) International Publication Date  
14 June 2001 (14.06.2001)

PCT

(10) International Publication Number  
WO 01/42881 A2

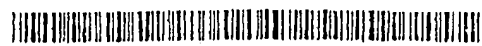
- (51) International Patent Classification?: G06F  
(21) International Application Number: PCT/US00/42665  
(22) International Filing Date: 6 December 2000 (06.12.2000)  
(25) Filing Language: English  
(26) Publication Language: English  
(30) Priority Data:  
60/169,101 6 December 1999 (06.12.1999) US  
Not furnished 5 December 2000 (05.12.2000) US  
(71) Applicant: B-BOP ASSOCIATES, INC. [US/US]:  
Suite 100, One Bay Plaza, 1350 Old Bayshore Highway,  
Burlingame, CA 94010 (US).  
(72) Inventors: DODDS, David; 16 Old Barn Road, Stamford,  
CT 06905 (US). KUO, Larry; 120 Morning Star Drive,  
San Jose, CA 95131 (US). SENGUPTA, Soumitra; 15  
First Street, Apt. 5, Stamford, CT 06905 (US). LINDSEY,  
Bill; 2203 Hastings Drive, Apt. 28, Belmont, CA 94002  
(US).  
(74) Agent: LOHSE, Timothy, W.; Gray Cary Ware & Frei-  
denrich LLP, 400 Hamilton Avenue, Palo Alto, CA 94301-  
1825 (US).  
(81) Designated States (national): AE, AL, AM, AT, AU, AZ,  
BA, BB, BG, BR, BY, CA, CH, CN, CR, CU, CZ, DE, DK,  
DM, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL,  
IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU,  
LV, MA, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT,  
RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA,  
UG, UZ, VN, YU, ZA, ZW.  
(84) Designated States (regional): ARIPO patent (GH, GM,  
KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian  
patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European  
patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE,  
IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF,  
CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

(54) Title: SYSTEM AND METHOD FOR THE STORAGE, INDEXING AND RETRIEVAL OF XML DOCUMENTS USING RELATIONAL DATABASES



BEST AVAILABLE COPY

(57) **Abstract:** A system and method for assigning attributes to XML document nodes to facilitate their storage in relational databases and the subsequent retrieval and reconstruction of pertinent nodes and fragments in original document order is provided. Since these queries are performed using relational database query engines, the speed of their execution is significantly faster than that using more exotic systems such as object-oriented databases. Furthermore, this method is portable across all vendor platforms, and so can be deployed at client sites without additional investments in database software.



**Published:**

— Without international search report and to be republished upon receipt of that report.

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

SYSTEM AND METHOD FOR THE STORAGE, INDEXING AND RETRIEVAL  
OF XML DOCUMENTS USING RELATIONAL DATABASES

Priority Claim

This application claims priority under 35 USC §§ 119 and 120 from US  
5 Provisional Patent Application No. 60/169,101 filed December 6, 1999.

Background of the Invention

This invention relates generally to a system and method for storing documents  
in one format in a database having a different format and in particular to a system and  
method for storing and retrieving eXtensible Markup Language (XML) documents  
10 using a relational database.

The new eXtensible Markup Language (XML) protocol is poised to become the  
lingua franca of the Internet for capturing and electronically transmitting information.  
The advantage of XML, as compared to the older hypertext markup language protocol  
(HTML), is that it contains tags which render semantic significance to the information  
15 between the tags (e.g., the text between the tags is the last name of an author). In  
contrast, HTML tags are used primarily for specifying how the information is to be  
displayed in a browser (e.g., show the text between the tags in bold Arial font).  
Additionally, using known eXtensible Stylesheets (written in XSL), one may specify  
not only the format of how different XML elements are to be shown in a browser, but  
20 also the order in which they are to be displayed. These features of XML give a user  
much greater power and flexibility in searching for relevant information since a search  
may be performed using the tags that contain the semantic information. In addition,  
XML permits examining the information from different perspectives once it is found  
by the user.

To take full advantage of the possibilities that the XML protocol affords, it is desirable to devise an efficient means of storing, indexing and retrieving (via queries) XML documents. Typical RDMS, ODMS and flat files are slow and inefficient at  
 5 storing XML documents. A preferred way of building Document Object Model (DOM) representations of the XML documents and then traversing the resulting trees to locate relevant nodes is only acceptable for small documents since memory becomes a limiting factor when the XML documents approach even moderate sizes. In addition, searches are not optimal since all searches must begin at the root of the document  
 10 instead of at any node in the document. Moreover, it is not possible to search across a collection of documents (e.g. poems, novels, short stories and plays) for a particular character or the author.

At the same time, XML documents present unique challenges to storage in relational databases since their semi-structured nature often leads to a proliferation of  
 15 tables when normalization is carried out. Given that relational database technology has seen great strides over the past couple of decades, it would be desirable and useful to provide a clean way of representing XML documents in relational terms. It is therefore the goal of the present invention to provide a system and method for the storage, indexing and retrieval of XML documents using relational databases.

## 20 Summary of the Invention

A system and method for storing, indexing and retrieving XML documents in a relational database is provided in accordance with the invention. The method may include identifying and assigning properties and encodings to the nodes of an XML document that will make them amenable to storage and retrieval using relational

databases. The method has several advantages. It allows the system to capture and reproduce the structure of not only the whole document, but fragments of each document as well. It also permits a user to traverse the XML tree, figuratively, by means of string manipulation queries instead of following pointers in memory or  
5 computing joins between tables, which are computationally more expensive operations. Finally, the properties and encodings that are attached to the nodes are compact and can be effectively indexed, thus enhancing the performance of queries against the database.

The system in accordance with the invention uses any relational database  
10 management system to store the XML documents so that the system and method are not dependent on any particular relational database implementation. The system permits a user to search through the XML documents stored in the relational database from any node element without starting from the root element of the document. This provides optimal efficiency during search and retrieval that can not be obtained using  
15 other methods today. In addition, a document may be constructed from any node and its descendants. The system also permits documents conforming to any XML schema to be stored in an efficient manner. The system can also store any well formed XML document that do not conform to any schema or DTD (Document Type Definition). This is an important feature as a large majority of XML documents generated do not  
20 conform to a schema or DTD.

In accordance with the invention, the system may include a converter and a searcher that permit XML documents to be stored in the relational database and retrieved from a relational database using typical SQL queries. In a preferred embodiment, the converter and searcher may be one or more software modules being executed by a  
25 central processing unit on a computer system. In accordance with the invention, the method for storing the XML documents may include the steps of generating an

XMLName value for each element in the document tree, generating a NamePath value for each node of the document and generating an OrderPath value for each node of the document. Collectively, assigning values to these elements are called encodings.

These encodings result in efficient storage, indexing and searching of XML documents without destroying the underlying hierarchical structure of the documents. The retrieval of the XML documents once they are in the relational database is relatively easy since typical string matching SQL queries may be used.

Thus, in accordance with the invention, a computer system and method for manipulating an XML document using a relational database is provided. The system comprises a converter that receives an XML document and generates a set relational database tables based on the hierarchical structure of XML a database for storing the relational database tables, and a searcher for querying the generated relational database table in the database to locate content originally in the XML document that is now stored in the relational database tables wherein the located content is returned to the user as an XML document or a portion of an XML document as desired by the user which can be another software module. The invention also includes the searcher that can convert queries specified on the XML document or document collections and convert them to simple SQL queries to retrieve the content desired by the user.

In accordance with another aspect of the invention, a computer system for storing an XML document using a relational database is provided wherein the system comprises a converter that receives an XML document and generates relational database tables based on the structure of the XML document. The converter further comprises a software module that generates a unique name attribute for each node in the XML document, a software module that generates a path attribute for a particular node of the XML document wherein the path attribute comprises a list of the name attributes for the one or more nodes from the particular node to a root node of the XML document, a software module that generates an order attribute for the particular node,

the order attribute comprising an enumerated order of the particular node from the root node to the particular node, and a software module that generates a NodeValue attribute containing a value of the particular node. Collectively these attributes are called encodings that result in efficient storage, indexing and searching of XML documents without destroying the underlying hierarchical structure of the documents.

In accordance with yet another aspect of the invention, a data structure that stores a node of interest of an XML document in a relational database is provided. The data structure comprises an XMLName attribute comprising a unique name for the node of interest, a NamePath attribute comprising a list of the XMLName attributes for the one or more nodes from the node of interest to a root node of the XML document, an OrderPath attribute comprising an enumerated order of the node of interest from the root node to the node of interest, and a NodeValue attribute containing a value of the node of interest. Collectively these attributes are called encodings that result in efficient storage, indexing and searching of XML documents without destroying the underlying hierarchical structure of the documents.

#### Brief Description of the Drawings

Figure 1 is a diagram illustrating a personal computer implementation of an XML document storage and retrieval system in accordance with the invention;

Figure 2 is a diagram illustrating more details of the XML document storage and retrieval system in accordance with the invention;

Figure 3 is a diagram illustrating an example of a document type definition (DTD) tree for an XML document;

Figure 4 is a diagram illustrating an XML document corresponding to the table shown in Figure 3;

Figure 5 is a flowchart illustrating an example of a method for storing XML documents in a relational database in accordance with the invention; and

Figure 6 is a flowchart illustrating a method for retrieving an XML document from a search of a relational database in accordance with the invention.

## 5 Detailed Description of a Preferred Embodiment

The invention is particularly applicable to a software implemented XML document storage and retrieval system and method and it is in this context that the invention will be described. It will be appreciated, however, that the system and method in accordance with the invention has greater utility since it may be  
10 implemented in hardware instead of software.

Figure 1 is a block diagram illustrating an embodiment of a software-based XML document storage and retrieval system 20 in accordance with the invention. In this embodiment, the storage and retrieval system 20 may be executed by a computer 22. The computer 22 may be a typical stand-alone personal computer, a computer  
15 connected to a network, a client computer connected to a server or any other suitable computer system. For purposes of illustration only, an embodiment using a stand-alone computer 22 will be described herein.

The computer 22 may include a central processing unit (CPU) 28, a memory 30, a persistent storage device 32, such as a hard disk drive, a tape drive, an optical  
20 drive or the like and a storage and retrieval system 34. In a preferred embodiment, the storage and retrieval system may be one or more software applications stored in the persistent storage device 32 of the computer that may be loaded into the memory 30 so that the storage and/or retrieval functionality of the storage and retrieval system may be executed by the CPU 28. The computer 22 may be connected to a remote server or

other computer networks that permit the computer 22 to network with and share the stored XML document with other computers or to perform searches on XML stored documents on other computer systems.

The computer 22 may further include one or more input devices 36, such as a  
5 keyboard 38, a mouse 40, a joystick or the like, a display 42 such as a typical cathode ray tube, a flat panel display or the like and one or more output devices (not shown) such as a printer for producing printed output of the search results. The input and output devices permit a user of the computer to interact with the storage and retrieval system so that the user may, for example, enter a query using the input devices and  
10 view the results of the query on the display or print the query results.

As described below in more detail, the storage and retrieval system 34 may include one or more different software modules that provide XML document storage capabilities and XML document retrieval capabilities in accordance with the invention. Now, more details of the storage and retrieval system will be described.

15 Figure 2 is a diagram illustrating more details of the XML document storage and retrieval system 34 in accordance with the invention. The system may include a converter module 50, a searcher module 52 and a relational database 54. Each of the modules may be implemented, in a preferred embodiment, as a software application being executed by a CPU as described above. The relational database 54 may be any  
20 type of relational database so that the system 34 in accordance with the invention may be used to store XML documents in any relational database system.

The converter module 50 accepts XML documents, processes them and outputs relational data about the XML documents as described below that is stored in the typical relational database 54. The searcher module 52 generates a user interface to a  
25 user, permits the user to enter a text string type relational database query, processes the

query by communicating a query to the relational database 54 and sends the results of the query in its original XML form to the user so that the user may view or print the query results. In combination, the two modules shown permit XML documents to be stored in any relational database system and then permits a user to enter a typical text string relational database query in order to retrieve XML documents from the relational database that match the text string query. Each of these modules will be described in more detail below. Now, an example of a Document Type Definition (DTD) of an XML document will be described to better understand the invention. This example of the DTD will be used as an example to illustrate the storage and retrieval system in accordance with the invention.

Figure 3 is a diagram illustrating an example of a Document Type Definition (DTD) tree 60 for an XML document. Although not required to do so, an XML document typically conforms to a DTD which, loosely speaking, is a schema for the data found in the document. However, XML documents are semi-structured in the sense that there are elements specified in the DTD that may be optionally present and some that may be present more than once. This is in contrast to typical relational database tables where each record must have either zero (if it is NULL) or only one value for an attribute.

XML documents also resemble an object-oriented database in that there are parent-child relationships between elements which are not found between attributes in a relational database. The following example of an XML document should help make these distinctions more clear. An example of the XML DTD syntax may be:

```
<!ELEMENT library (book*, periodical*)>
25 <!ELEMENT book (title, author+)>
    <!ATTLIST book edition CDATA #REQUIRED>
```

<ELEMENT author (title?, firstname, lastname)>

In the above example, elements that appear within parentheses are the children of elements before the parentheses. In addition a "\*" denotes 0 or more occurrences of the element, a "+" denotes one or more occurrences and a "?" denotes 0 or 1 occurrence. The above example DTD may be represented by the DTD tree shown in Figure 3. The DTD tree 60 may include a root node 62 (containing the element "library" in this example), one or more intermediate nodes 64 and one or more leaf nodes 66 that do not have any further nodes attached to them. An example of an XML document 70 that conforms to the DTD is shown in Figure 4. It contains the instances of elements in the DTD tree along with data for each element. The conversion of this example of an XML document into a format that may be stored in a relational database in accordance with the invention will now be described.

Figure 5 is a flowchart illustrating an example of a method 80 for storing XML documents in a relational database in accordance with the invention. The method involves computing three properties, each of which is described below, for each XML document node so that the XML document may be stored, in an efficient manner, in a relational database. The encoding scheme set forth below is a preferred encoding embodiment. However, other encoding schemes may also be used. For example, the encoding set forth below (e.g., 1/2/5/6) may be represented as 1 raised to the power 1, 2 raised to the power 2, 3 raised to the power 5 and 4 raised to the power 6 and so on. That way, instead of performing string manipulation, the system would be doing factorization. Based on this other encoding, the factorization approach can generate faster queries and save indexing and database space. Thus, the invention is not limited to any particular encoding and the encodings in accordance with the invention are

created based on the structure of the document and then the encodings are used to store, index and search for the content while preserving the hierarchy of the document.

In a first step 81 of the method, it is determined if an element is ready for processing.

If there is an element ready for processing, then the method generates an XMLName

5 property for the particular element. If an element is not ready for processing, but an attribute of the XML document is read for processing, then the method also generates the XMLName property for the particular attribute. In more detail, the method starts by assigning each element name a unique XMLName property (in this example, the property is alphanumeric). For the example above, we could assign the XMLNames as  
10 shown in Table 1 (the XMLName Table).

Table 1 (the "XMLName Table")

Element or Attribute Name	XMLName
library	1
book	2
periodical	3
edition	4
title	5
author	6
firstname	7
lastname	8

Note that "title" gets only one XMLName value even though the element appears twice in the DTD tree as either the title of a book or the title of an author. This  
15 allows for more XMLName attributes to be encoded given strings of a specific length.

Now, in step 84, a NamePath value is automatically determined for each node of the DTD tree. In particular, the NamePath value may be constructed from the XMLNames of each node on the path from the root node to the node of interest. From this analysis, we obtain the following table of NamePath values for the example XML

5 document:

NamePath Table

DTD Node	NamePath
library	1
library/book	1/2
library/periodical	1/3
library/book/edition	1/2/4
library/book/title	1/2/5
library/book/author	1/2/6
library/book/author/title	1/2/6/5
library/book/author/firstname	1/2/6/7
library/book/author/lastname	1/2/6/8

As shown in the table, each DTD node, such as "library/book/author/lastname", has a corresponding NamePath value, such as "1/2/6/8". In this manner, using the NamePath values, it is possible to navigate through the XML document using the relational database. In other words, using this table, the path to any node in the DTD tree (and hence the XML document) may be easily determined. This table may also be stored in the relational database.

Next, in step 86, the method may automatically generate an OrderPath value for each node in the XML document. In particular, each number in the slash-separated OrderPath (see the table below) denotes the breadth-wise enumerated order of the node on the path from the root to the node of interest. Each document node may also inherit the NamePath of the DTD node of which it is an instance. A full DocNode Table for the example XML document looks like this:

DocNode Table

NodeName	NamePath	OrderPath	NodeValue
library	1	1	
book	1/2	1/1	
edition	1/2/4	1/1/1	first
title	1/2/5	1/1/2	The XML Revolution
author	1/2/6	1/1/3	
title	1/2/6/5	1/1/3/1	Software Engineer
firstname	1/2/6/7	1/1/3/2	David
lastname	1/2/6/8	1/1/3/3	Hollenbeck
author	1/2/6	1/1/4	
title	1/2/6/5	1/1/4/1	Chief Architect
firstname	1/2/6/7	1/1/4/2	Carol
lastname	1/2/6/8	1/1/4/3	Bohr
book	1/2	1/2	
edition	1/2/4	1/2/1	second
title	1/2/5	1/2/2	Java Classes for XML
author	1/2/6	1/2/3	
firstname	1/2/6/7	1/2/3/1	Carol
lastname	1/2/6/8	1/2/3/2	Hollenbeck
author	1/2/6	1/2/4	
title	1/2/6/5	1/2/4/1	XML Guru
firstname	1/2/6/7	1/2/4/2	David
lastname	1/2/6/8	1/2/4/3	Bohr

- As shown in the Table that may be stored in a relational database, each document node may include a NodeName value (the name of the element), a
- 5 NamePath value (See above), an OrderPath Value (automatically generated during this step), and a NodeValue value (containing the actual data in that particular node).

In step 88, the method determines if there are any more nodes to process and loops back to step 81 if there are more nodes. If all of the nodes have been processed, then the DocNode Table may be saved in the relational database. In this manner, an XML document is automatically processed in order to generate a DocNode Table that  
 5 may be stored in any relational database. Once the DocNode table is generated by the system, it may be searched as will now be described in more detail.

Figure 6 is a flowchart illustrating a method 100 for retrieving an XML document from a search of a relational database in accordance with the invention. In step 102, the user or the system using user input, may generate a relational database  
 10 query. In step 104, the system may query the relational database and in step 106, the query results are output to the user. In accordance with the invention, the system may convert the query results back into references to portions of the XML document so that the user may review the portions of the XML document retrieved during the search in step 108. Now, several examples of retrieving XML documents based on a relational  
 15 database search will be provided. In particular, a few examples will be shown of how the system may use the NamePath and OrderPath values to select nodes with desired attributes from the XML document repository and also may construct fragments of the original XML documents containing these selected nodes. In all the sample queries below, we assume that we know the context (i.e., the position within the DTD tree) of  
 20 the nodes we are interested in.

In a first example, a user wants to query the XML document repository to return the titles of all books who have an author with the title of "Chief Architect". Since we know the context of title (i.e., library/book/author/title), we can consult the XMLName Table to obtain the relevant XMLNames and construct the NamePath of  
 25 title which is "1/2/6/5" in this example. Then, the system may issue the first query that is:

"Select OrderPath from DocNodeTable where NamePath = '1/2/6/5' and NodeValue = 'Chief Architect'"

This query returns an OrderPath of "1/1/4/1" as the result. Since we also know that the element "book" is a grand-parent of element "title", we can deduce that its  
5 OrderPath is 1/1. Finally we construct the NamePath of the element "book title" as "1/2/5" and execute the second query that is :

"Select NodeValue from DocNodeTable where NamePath = '1/2/5' and OrderPath like '1/1/%'".

This second query returns the value "The XML Revolution" as the result. This  
10 result accomplishes the user goal of returning all books whose author's title is "Chief Architect". In this manner, the XML document repository is queried using typical relational database queries.

In this second example, the user wants to search for the titles of all books who have an author by the name of Carol Hollenbeck. To accomplish this, the system may  
15 generate a first query to select the OrderPaths of all firstname nodes with the value Carol:

"Select OrderPath from DocNodeTable where NamePath = '1/2/6/7' and NodeValue = 'Carol'"

This query returns "1/1/4/2" and "1/2/3/1" as the result set. Next, a second  
20 query is generated to select the OrderPaths of all lastname nodes with the value Hollenbeck:

"Select OrderPath from DocNodeTable where NamePath = '1/2/6/8' and NodeValue = 'Hollenbeck'"

This query returns "1/1/3/3" and "1/2/3/2" as the result set. Since we know  
firstname and lastname nodes of the same person belong to the same parent author  
node, we can deduce from the result sets that only the nodes with OrderPaths "1/2/3/1"  
and "1/2/3/2" are of interest to us. Thus, we want the title of the book with OrderPath  
5 1/2, which we can retrieve with the following query:

"Select NodeValue from DocNodeTable where NamePath = '1/2/5' and  
OrderPath like '1/2/%'"

This query returns "Java Classes for XML" as the result which is the proper  
result.

10 In a third example, the user wants to be returned all the information pertaining  
to the authors of "The XML Revolution" and presented in the original document order.  
Thus, first, the OrderPath of the relevant title node is determined by the following  
query:

"Select OrderPath from DocNodeTable where NamePath = '1/2/5' and  
15 NodeValue = 'The XML Revolution'"

This query returns "1/1/2" as the result. Thus, as a result of the first query, we  
know that the OrderPath of the relevant book node is "1/1". Since the nodes for all  
author information are descendants of the author node (that has NamePath "1/2/6"),  
which in turn is a child of the "book" node, we can execute the following query to  
20 obtain the required result:

"Select NodeValue from DocNodeTable where NamePath like '1/2/6/%' and  
OrderPath like '1/1/%' Order by OrderPath"

This query returns "Software Engineer, David, Hollenbeck, Chief Architect, Carol, Bohr" in the original document order as the result set.

Now, several enhancements to the system and method described above will be provided. In accordance with another aspect of the invention, the XMLName Table  
 5 may be cached in memory. In particular, to facilitate construction of the NamePath values, we can store the contents of XMLName Table in a hash table which we keep resident in memory. This prevents the execution of multiple queries against the database to obtain all the necessary XMLName values. In accordance with yet another aspect of the invention, the XMLName values may be divided into NameSpaces. In  
 10 particular, as the number of XMLName values increases, it may become necessary to divide the values into various namespaces to keep the lengths of the names short. XMLName values from namespaces relevant for working with a particular document can then be brought into the cache when necessary without having to bring the entire XMLNameTable into memory.

15 In accordance with yet another aspect of the invention, the system may use base-64 encoding. In particular, to reduce the amount of storage required for the XMLName, NamePath, and OrderPath tables in the relational database, we could consider using a Base-64 encoding scheme instead of alphanumeric strings. In accordance with the invention, it is also possible to add a DigitPath attribute as an  
 20 adjunct attribute to OrderPath so that the system can ensure proper sorting of nodes while obviating the need for place-holding characters as the number of characters increases. For example, to sort the paths "1/10/2" and "1/2/3" properly, the system would have needed to encode the second as "1/-2/3". However, if we added "1/2/1" and "1/1/1" as DigitPaths and ordered the results by these before OrderPaths, then we  
 25 would be able to do without the place-holding dashes.

In accordance with the invention, a ReverseNamePath attribute may be automatically generated to further improve the speed of queries. In particular, since it is possible to have an XML document that is an instance of a DTD sub-tree, we may need to evaluate an expression such as:

5 "Select NodeValue from DocNode Table where NamePath like '%/1/2/3'"

Since indexes built on NamePath generally do not help in the execution of such queries, we can improve performance by having a ReverseNamePath attribute constructed by reversing the order of the XMLNames in the path expression. Thus, in accordance with the invention, the above query would now read:

10 "Select NodeValue from DocNodeTable where ReverseNamePath like '3/2/1/%'"

In accordance with the invention, the system may include a transformation engine that converts XPath expressions into equivalent SQL statements involving NamePath and OrderPath attributes so that the converted queries would then be  
15 executed against the repository.

In summary, a system and method for assigning attributes to XML document nodes to facilitate their storage and indexing in relational databases and the subsequent retrieval and re-construction of pertinent nodes and fragments in original document order is provided. Since these queries are performed using relational database query  
20 engines, the speed of their execution is significantly faster than that using more exotic systems such as object-oriented databases. Furthermore, this method is portable across all vendor platforms, and so can be deployed at client sites without additional investments in database software.

In accordance with the invention, the hierarchical relationships of XML documents are encoded so that the XML documents may be mapped to a set of relational tables. Once the mapping and encoding is completed, then searching and querying of the XML documents may be done by mapping any XML query language  
5 (which is well known) to SQL (also well known) automatically.

While the foregoing has been with reference to a particular embodiment of the invention, it will be appreciated by those skilled in the art that changes in this embodiment may be made without departing from the principles and spirit of the invention as set forth in the appended claims.

CLAIMS:

1           1.       A computer system for manipulating an XML document using a  
2       relational database, comprising:  
3           a converter that receives an XML document and generates a pre-determined set  
4       of relational database tables based on the XML document;  
5           a database for storing the relational database table; and  
6           a searcher for querying the generated relational database table in the database to  
7       locate content originally in the XML document that is now stored in the relational  
8       database table wherein the located content is returned to the user as a portion of an  
9       XML document.

1           2.       The system of Claim 1, wherein the converter further comprises a  
2       software module that generates a unique name attribute for each node in the XML  
3       document.

1           3.       The system of Claim 2, wherein the converter further comprises a  
2       software module that generates a path attribute for a particular node of the XML  
3       document wherein the path attribute comprises a list of the name attributes for the one  
4       or more nodes from the particular node to a root node of the XML document.

1           4.       The system of Claim 3, wherein the converter further comprises a  
2       software module that generates an order attribute for the particular node, the order  
3       attribute comprising an enumerated order of the particular node from the root node to  
4       the particular node.

1           5.     The system of Claim 4, wherein the converter further comprises a  
2     software module that generates a NodeValue attribute containing a value of the  
3     particular node.

1           6.     The system of Claim 5, wherein the searcher further comprises a query  
2     generator that generates a query into the database to find a piece of information in the  
3     database  
4                 corresponding to information in a node of the XML document and a converter  
5     that converts the results of the query into portions of an XML document that are  
6     displayed to the user.

1           7.     The system of Claim 2, wherein the name attribute for each node in the  
2     XML document is stored in a hash table so that the name attributes are retrieved from  
3     the hash table instead of the database.

1           8.     The system of Claim 2, wherein the name attributes of the nodes of the  
2     XML document are divided into one or more categories so that related name attributes  
3     are grouped together.

1           9.     The system of Claim 1, wherein the name attributes are encoded using  
2     base-64 encoding.

1           10.    The system of Claim 3, wherein the converter further comprises a  
2     software module that generates a reverse path comprising the list of name attributes  
3     from the path attribute in reverse order.

1           11. The system of Claim 1, wherein the converter further comprises a  
2 transform engine that converts Xpath expressions in the XML document into SQL  
3 queries.

1 12. A computer system for storing an XML document using a relational  
2 database, comprising:  
3 a converter that receives an XML document and generates a relational database  
4 table based on the XML document;  
5 the converter further comprising a software module that generates a unique  
6 name attribute for each node in the XML document, a software module that generates a  
7 path attribute for a particular node of the XML document wherein the path attribute  
8 comprises a list of the name attributes for the one or more nodes from the particular  
9 node to a root node of the XML document, a software module that generates an order  
10 attribute for the particular node, the order attribute comprising an enumerated order of  
11 the particular node from the root node to the particular node, and a software module  
12 that generates a NodeValue attribute containing a value of the particular node.

1           13.     A method for manipulating an XML document using a relational  
2     database, comprising:  
3             generating a relational database table based on an XML document wherein the  
4     information about each node of the XML document is stored in a row of the table;  
5             storing the relational database table in a database; and  
6             querying the generated relational database table in the database to locate  
7     content originally in the XML document that is now stored in the relational database  
8     table wherein the located content is returned to the user as a portion of an XML  
9     document.

1           14.    The method of Claim 13, wherein generating the table further comprises  
2   generating a unique name attribute for each node in the XML document.

1           15.    The method of Claim 14, wherein generating the table further comprises  
2   generating a path attribute for a particular node of the XML document wherein the path  
3   attribute comprises a list of the name attributes for the one or more nodes from the  
4   particular node to a root node of the XML document.

1           16.    The method of Claim 15, wherein generating the table further comprises  
2   generating an order attribute for the particular node, the order attribute comprising an  
3   enumerated order of the particular node from the root node to the particular node.

1           17.    The method of Claim 16, wherein generating the table further comprises  
2   generating a NodeValue attribute containing a value of the particular node.

1           18.    The method of Claim 17, wherein querying the database further  
2   comprises generating a query into the database to find a piece of information in the  
3   database corresponding to information in a node of the XML document and converting  
4   the results of the query into portions of an XML document that are displayed to the  
5   user.

1           19.    The method of Claim 14 further comprising retrieving the name  
2   attribute for each node in the XML document from a hash table so that the name  
3   attributes are retrieved from the hash table instead of the database.

1           20.    The method of Claim 14, wherein the name attributes of the nodes of  
2   the XML document are divided into one or more categories so that related name  
3   attributes are grouped together.

1           21.    The method of Claim 13, wherein the name attributes are encoded using  
2   base-64 encoding.

1           22.    The method of Claim 15, wherein generating the table further comprises  
2   generating a reverse path comprising the list of name attributes from the path attribute  
3   in reverse order.

1           23.    The method of Claim 13, wherein generating the table further comprises  
2   converting Xpath expressions in the XML document into SQL queries.

1           24.    A data structure that stores a node of interest of an XML document in a  
2   relational database, the data structure comprising:  
3       an XMLName attribute comprising a unique name for the node of interest;  
4       a NamePath attribute comprising a list of the XMLName attributes for the one  
5   or more nodes from the node of interest to a root node of the XML document;  
6       an OrderPath attribute comprising an enumerated order of the node of interest  
7   from the root node to the node of interest; and  
8       a NodeValue attribute containing a value of the node of interest.

1           25.    The data structure of Claim 24, wherein the data structure comprises a  
2   table in a relational database and each attribute comprises a column in the table in the  
3   relational database.

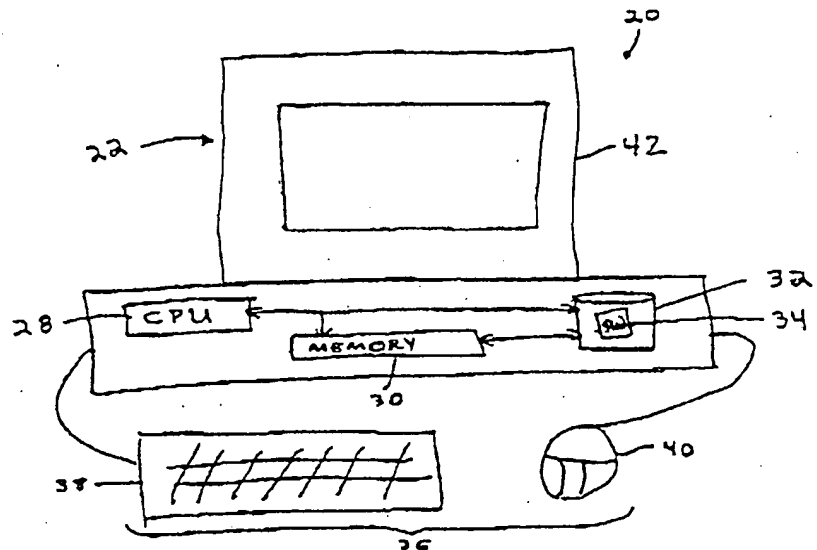


FIGURE 1

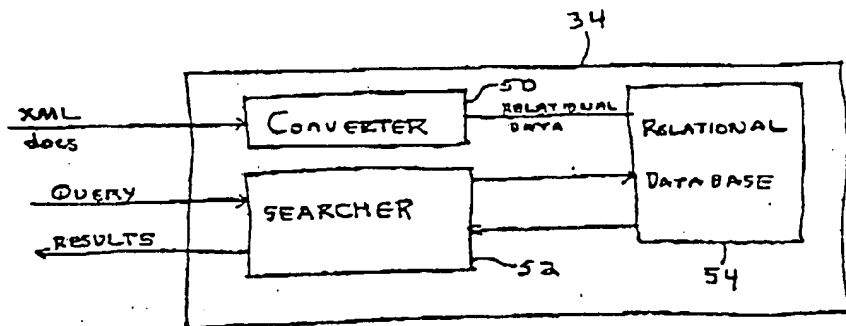


FIGURE 2

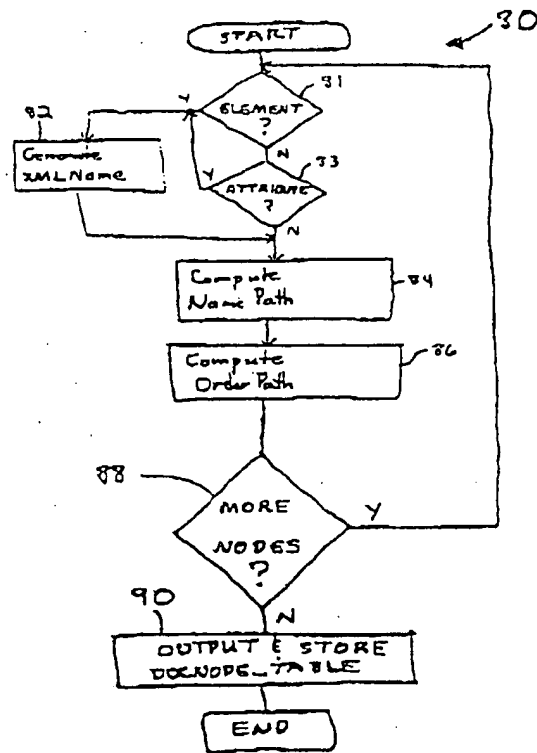


FIGURE 5

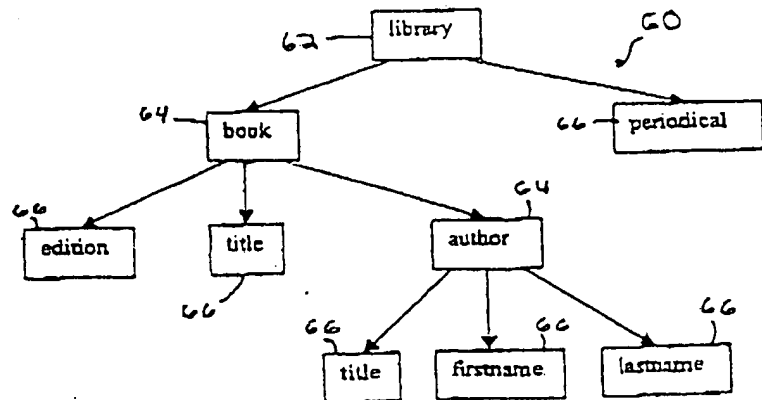


FIGURE 3

```

<library>
  <book edition='first'>
    <title>The XML Revolution</title>
    <author>
      <title>Software Engineer</title>
      <firstname>David</firstname>
      <lastname>Hollenbeck</lastname>
    </author>
    <author>
      <title>Chief Architect</title>
      <firstname>Carol</firstname>
      <lastname>Bohr</lastname>
    </author>
  </book>
  <book edition='second'>
    <title>Java Classes for XML</title>
    <author>
      <firstname>Carol</firstname>
      <lastname>Hollenbeck</lastname>
    </author>
    <author>
      <title>XML Guru</title>
      <firstname>David</firstname>
      <lastname>Bohr</lastname>
    </author>
  </book>
</library>

```

70

FIGURE 4

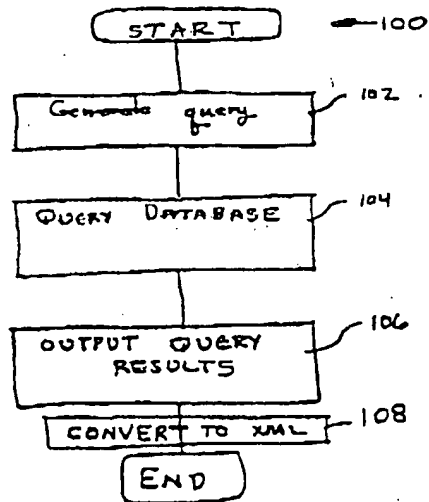


FIGURE 6

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization  
International Bureau



(43) International Publication Date  
16 August 2001 (16.08.2001)

PCT

(10) International Publication Number  
WO 01/59602 A1

(51) International Patent Classification<sup>7</sup>: G06F 17/00

(21) International Application Number: PCT/US01/04698

(22) International Filing Date: 12 February 2001 (12.02.2001)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:  
60/182,047 11 February 2000 (11.02.2000) US

(71) Applicant (for all designated States except US): ACTA  
TECHNOLOGIES, INC. [US/US]; 1667 Plymouth  
Street, Mountain View, CA 94043 (US).

(72) Inventors; and

(75) Inventors/Applicants (for US only): GORELIK,  
Alexander [US/US]; 44153 Boitano Drive, Fremont, CA  
94539 (US). CHAWLA, Sachinder, Singh [US/US]; 1963  
Pine Street, San Francisco, CA 94109 (US). SYED, Awez,

Imran [US/US]; 10810 Brewer House Road, Rockville,  
MD 20852 (US). BURDA, Leon [US/US]; 22206 Quin-  
terno Court, Cupertino, CA 95014 (US). YEE, Mon,  
For [US/US]; 255 Yerba Buena Avenue, San Francisco,  
CA 94127 (US). GANTIMAHAPATRUNI, Sridhar  
[US/US]; 361 Sunset Avenue, Sunnyvale, CA 94086 (US).

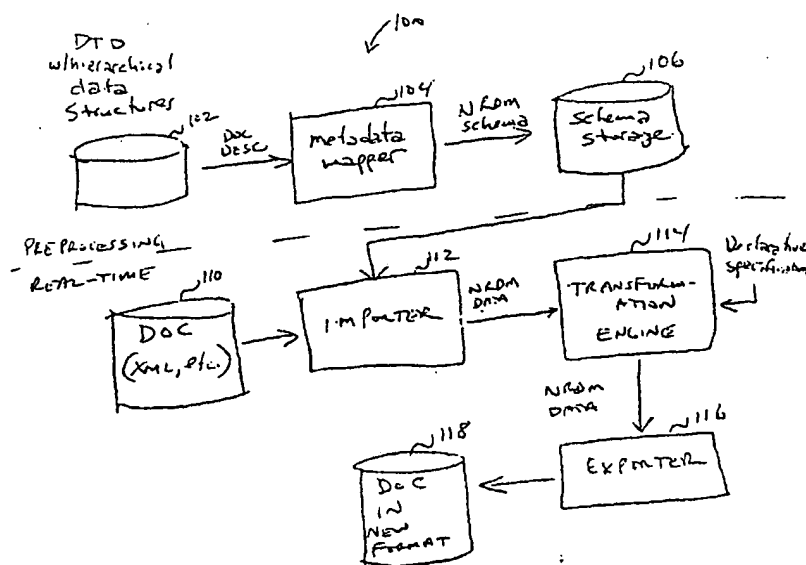
(74) Agents: ALBERT, Philip, H. et al.; Townsend and  
Townsend and Crew LLP, Two Embarcadero Center,  
Eighth Floor, San Francisco, CA 94111-3834 (US).

(81) Designated States (national): AE, AG, AL, AM, AT, AU,  
AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CR, CU, CZ,  
DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR,  
HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR,  
LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ,  
NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM,  
TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW.

(84) Designated States (regional): ARIPO patent (GH, GM,  
KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian  
patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European  
patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE,

[Continued on next page]

(54) Title: NESTED RELATIONAL DATA MODEL



(57) Abstract: A data base system (100) handles nested relational data model (NRDM) data. A metadata mapper (104) converts DTD w/hierarchical structures (102) to NRDM schema that are then stored in schema storage (106). A document (110) is passed to an importer (112), then to a transformation engine (114) and an exporter (116) to result in a document in a new format (118). Because the transformation engine (114) operates on NRDM structures, the transformations can be expressed as a declarative specification, thus simplifying the process of transforming complex data. Exporter (116) exports the data in a suitable form such as XML documents, relational tables or flat files.

WO 01/59602 A1

WO 01/59602 A1



IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

*For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.*

**Published:**

— with international search report

## NESTED RELATIONAL DATA MODEL

### COPYRIGHT NOTICE

A portion of the disclosure of this patent document contains material that is subject to copyright protection. The copyright owner has no objection to the facsimile reproduction by anyone of the patent document or the patent disclosure as it appears in the Patent and Trademark Office patent file or records, but otherwise reserves all copyright rights whatsoever.

### FIELD OF THE INVENTION

The present invention relates to information management in general and more particularly to methods for using Nested Relational Data Models (NRDMs) to manage information.

### BACKGROUND OF THE INVENTION

Information is commonly managed in units of documents. For example, sales, distribution and manufacturing information might be contained within documents such as sales invoices or orders. Increasingly, documents pass between parties in electronic form, in a process generally referred to as EDI (Electronic Data Interchange). In electronic form, the documents are not limited to the text and images shown on the printed page, but can include formatting and "metadata" (data about the data). One example of a format for an electronic document that contains metadata is the Extended Markup Language (XML).

Several products on the market allow mapping of XML documents to SQL tables or vice versa and several products on the market allow mapping of EDI documents to relational tables or vice versa, but these products typically require procedural specifications of how to perform the conversion, such as programming code. Traditional Relational Database Management Systems (RDMS's) such as described by Date or Ullman or implemented by Oracle, IBM, Microsoft and others as well as distributed databases as described in Ceri or U.S. Patent Nos. 5,884,310 and 5,596,744, implement declarative transformations of relational data.

A class of systems called intelligent gateways (such as Sybase's OmniServer system) allow declarative rules to be transparently applied to heterogeneous relational databases. Another class of systems called Replication Servers (such as

described by U.S. Patent No. 5,737,601 or implemented as Sybase's Replication Server, Oracle's Replication Server, or the like) can provide homogeneous or heterogeneous data replication.

Additional class of systems called the ETL (Extraction, Transformation, Loading) systems such as Microsoft DTS, Informatica PowerMart and D2K Tapestry provide extraction, transformation and loading of heterogeneous data between relational database systems. Some of these products support converting hierarchical files into a relational form by "flattening" the hierarchical files, making multiple passes through a hierarchical file and, at each pass, pulling out different parts of the hierarchy.

Yet another class of systems that address mapping of relational data to a programming object, as exemplified by U.S. Patent Nos. 6,175,837, 6,163,781, 6,134,559, 5,907,846, 5,873,093, 5,832,498, or products from Persistence, Bea and others. This class of tools maps persistently stored relational data to an object-oriented memory representation as well as mapping the data from an object-oriented memory representation to a set of persistent relational tables.

Another class of prior art exists that provides object-oriented access to non-relational databases, as described in U.S. Patent Nos. 5,799,313, 5,778,379, and 5,542,078. This class of systems addresses the mapping of data from hierarchical databases such as IMS, object oriented databases and relational databases to an object-oriented programming object or database.

Considerable research has been done on Nested Relational Data Models as described in \_\_\_, "Lecture Notes in Computer Science Volume 595: M. Levene – The Nested Universal Relation Database Model" and \_\_\_, "Lecture Notes in Computer Science Volume 361: S. Abiteboul et al. – Nested Relations and Complex Objects in Databases". That research focused mainly on defining the data model and specific operations on it.

It is known to graphically map disparate schemas to each other. See, for example, U.S. Patent Nos. 5,850,631 and 5,806,066. It is also known to map data between different structures. See for example, U.S. Patent Nos. 5,627,972 and 5,119,465.

## SUMMARY OF THE INVENTION

In one embodiment of data processing system according to the present invention, hierarchical documents or hierarchical messages are mapped to a Nested Relational Data Model to allow for transformation and manipulation using declarative

statements. The resulting nested data can be converted to a relational format and mapped to multiple relational tables, and/or converted from a nested relational format to an external hierarchical format, such as XML.

5 The system can specify and execute declarative rules to extract, transform, integrate, load and update hierarchical and relational data. The system can also be used for extending documents with relational and non-relational data and applying updates based on these documents to relational database targets. The system can also be used for mapping Nested Relational Data to function calls that accept tables as parameters and return multiple scalar and table parameters as output.

#### 10 BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 shows a table that is related to a single row of another table.

Fig. 2 shows the data of Fig. 1, organized as multiple rows in a single table.

Fig. 3 shows the data of Fig. 1, organized as multiple tables related by a

15 join.

Fig. 4 illustrates multiple levels of nested tables contained in one column.

Fig. 5 illustrates a more general example of multiple levels of nested tables contained in more than one column.

20 Fig. 6 is a block diagram of a database system according to one embodiment of the present invention.

Fig. 7 illustrates schema relating to nested tables; Fig. 7A shows input tables and Fig. 7B shows an output schema.

Fig. 8 illustrates a process of grouping values across nested tables.

25 Fig. 9 illustrates a process of unnesting data; Fig. 9A shows how a table with a nested table would be unnested into a cross-product of the parent table and a child (nested) table; Fig. 9B illustrates unnesting into separate tables; Fig. 9C illustrates unnesting at multiple levels.

Fig. 10 illustrates a case where unnesting might produce unintended effects.

30 Fig. 11 graphically illustrates an unnesting process and its effects on a query.

Fig. 12 illustrates a process of converting a DTD to tables.

Fig. 13 illustrates the XML encoding of a DTD definition.

Fig. 14 illustrates various real-time data flows.

Fig. 15 illustrates an operation of joining two inputs in a query.

Fig. 16 illustrates real-time data flows that use supplementary information.

Fig. 17 illustrates data flows depending on cached values.

Fig. 18 illustrates branching data flows based on rules.

Fig. 19 is an illustration of a complex real-time data flow.

Fig. 20 is an illustration of a GUI for specifying a data flow.

Fig. 21 is a block diagram of a schema conversion system.

Figs. 22-26 are tables illustrating various aspects of an NRDM system.

### DESCRIPTION OF THE SPECIFIC EMBODIMENTS

In a specific embodiment a Nested Relational Data Model (NRDM) is designed to support hierarchical and relational components used to represent business data. Business documents are typically hierarchical with multiple repeating sets. For example, an order contains a set of repeating line items. It may also have a set of customers associated with it.

Business documents used to exchange data between software systems within an enterprise or between enterprises need to be represented as complex hierarchical documents. The industry and the research community use well-known representations such as EDI and XML to capture and represent such documents. The system described herein provides methods for mapping such documents to a Nested Relational format, methods for transforming and manipulating of these documents represented using the Nested Relational Data Model, converting such documents to relational format and mapping them to multiple relational tables, and a method of converting the data in a nested relational format back to an external hierarchical format such as XML.

The system provides a method to apply declarative rules to map the hierarchical (e.g., XML or EDI) data to relational tables and vice versa; declarative rules to enrich hierarchical data with data from other relational or hierarchical sources; declarative rules to perform multi-stage transformations. The system allows declarative transformations to be applied to hierarchical data, and the ability to transparently apply rules to heterogeneous databases and files; as well as in the ability to apply multi-stage transformations. Declarative specifications (such as SQL) describe what to do with data, as opposed to procedural specifications (such as C++ code) that described how to do it.

"Nested data" is data in a table that is related to a single row of another table. Sales orders are often presented using nesting: the line items in a sales order are related to a single header. For a table of sales order headers, each row includes its own table of line items. An example of this is shown in Fig. 1. Of course, the same data could  
5 be represented without nested tables. For example, the data could be represented as multiple rows in a single table as shown in Fig. 2, or as multiple tables related by a join as shown in Fig. 3.

One source of data for a nested table is the result of a query using the values in the related row in the parent table. As used herein, "parent table" refers to a  
10 table within which another table is nested and "child table" or "nested table" refers to a table that is nested in a column of a parent table. A nested table is said to have a relationship with the table within which it is nested and where levels are associated with tables, a parent table would have a level that is designated with a number one higher than the child tables nested in that parent table. For example, Fig. 4 shows a parent table 10, a  
15 nested (child) table 12 one level below table 10 and nested tables 14(a)-(b) that are nested in table 12 and are two levels below table 10.

Preferably, a unique instance of each nested table exists for each row at each level of a relationship. As illustrated in Fig. 5, each row at each level can have any number of columns containing nested tables.

20 Fig. 6 shows various aspects of a database system 100 that handles NRDM data. System 100 is shown comprising a metadata mapper 104 that maps DTD 102 w/hierarchical structures to NRDM schema that are stored in schema storage 106. These components are shown as being part of a preprocessing section, with other portions being part of a real-time section, but it should be understood that all of the process or none of  
25 the processing might be done in real-time without departing from the essence of the invention. Notwithstanding that caveat, the descriptions below reference an example wherein DTDs are converted to NRDM schema and stored and documents are converted by system 100 in real-time after such conversion.

One such real-time process involved a document 110 being passed to an  
30 importer, then to a transformation engine (TE) 114 and an exporter 116 to result in a document in a new format 118 (in some cases, the formats of document 110 and document 118 might be the same, but some transformation has occurred). Document 110 is a structured document, such as an XML document, an HTML page, a document having other structure, or other structured data object.

Importer 112 converts the document into NRDM data so that TE 114 can operate on data in the NRDM space, thus simplifying many transform operations, as described below. TE 114 accepts data in NRDM format as its input and outputs data in NRDM format. Of course, data in NRDM (Nested Relational Data Model) format need not have nested data (for example, if the input data can be structured such that nesting is not needed). Because TE 114 operates on NRDM structures, the transformations performed by TE 114 can be expressed simply as a declarative specification, thus greatly simplifying the process of transforming complex data. In effect, importer 112 converts a hierarchical document into a relational database form to which declarative statements can be applied.

Exporter 116 exports the data in a suitable form, such as XML documents, relational tables or flat files.

#### Data Flows

In a graphical interface used to build data flows and/or nested data structures, such as the ActaWorks™ system developed by Acta, Inc. structures of nested data in input and output schemas of sources, targets, and transforms in data flows are presented to a designer. An example of an input schema 60 is shown in Fig. 7A and an example of an output schema 62 is shown in Fig. 7B. Input schema 60 shows a table A that has columns column1, column2 and a column for a nested table B, which in turn has columns column4 and column5. Input schema 60 also shows a table Z that has columns column11, column12 and a column for a nested table Y, which in turn has columns column14 and column15. In Fig. 7A, and others, nested tables appear with a table icon paired with a plus sign, which indicates that the object contains columns (a minus sign indicates that the object is open and if it has columns, those columns are visible).

In a relational database system (RDS) using a declarative language such as SQL, a query transform might take the form of a SELECT statement that is executed by the RDS. When working with nested data in an nested relational data model (NRDM) system according to some aspects of the present invention, the query can specify SELECTs at each level of a relationship defined in the output schema. Thus, while a SELECT statement might be constrained to include only references to relational data sets, a query that includes nested data might include a SELECT statement to define operations on each table in the output--each context for the input data set is transformed.

In such an NRDM system, the FROM clause descriptions and the behavior of the query are the same with nested data as with relational data, but the new interface of

contexts allows the data flow designer to distinguish multiple SELECTs from each other within a single query. At any context, the FROM clause can contain any top-level table from the input or any table that is a column of a table in the FROM clause of the next higher context.

When rows of one table (a child table) are nested inside another table (a parent table), the data set produced in the nested table is the result of a query against the first table using the related values from the second table. For example, if sales information is available as a header table and a line-item table, the sales information can be organized as a parent table of header information and a child table containing line-item data here the line-items are nested under the header table. The line items for a single row of the header table are equal to the results of a query including the order number, as might be found using the following statement:

```
SELECT * FROM LineItems
WHERE Header.OrderNo = LineItems.OrderNo
```

Correlation can be used to construct a nested table from columns from a higher-level context. In a nested-relational model, the columns in a nested table are implicitly related to the columns in the parent row. To take advantage of this relationship, the parent table can be used in the construction of the nested table. The higher-level column is a correlated column. Including a correlated column in a nested table may serve at least two purposes: 1) the correlated column is a key in the parent table and 2) making the correlated column an attribute in the parent table. Including the key in the nested table allows for the maintenance of you a relationship between the two tables after converting them from the nested data model to a relational model. Including the attribute in the nested table allows for the use of the attribute to simplify correlated queries against the nested data.

Correlated columns can include columns from the parent table and any other tables in the FROM clause of the parent table. If the correlated column comes from a table other than the immediate parent, the data in the nested table includes only the rows that match both the related values in the current row of the parent table and the value of the correlated column.

Values can be grouped across nested tables. Thus, when a statement includes a Group By clause for a table with a nested table, the grouping operation combines the nested tables for each group. For example, to assemble all the line items

included in all the orders for each state from a set of orders, the designer would set the Group By clause in the top-level of the data set to the state column (Order.State) and create an output table that includes State column (set to Order.State) and LineItems nested table. The result of such an operation might result with the table shown in Fig. 8. The result is a set of rows (one for each state) that has the State column and the LineItems nested table that contains all the LineItems for all the orders for that state.

Nested data can also be unnested. When loading a data set that contains nested tables into a relational (non-nested) target, the nested rows will be unnested. Take, for example, a message containing a sales order that uses a nested table to define the relationship between the order header and the order line items. To load the data into relational tables, the multi-level must be unnested. Unnesting a table produces a cross-product of the top-level table (parent) and the nested table (child), as shown in Fig. 9A. Different columns from different nesting levels might be loaded into different tables. A sales order, for example, may be flattened so that the order number is maintained separately with each line item and the header and line item information loaded into separate tables, as shown in Fig. 9B.

Any number of nested tables can be unnested at any depth. No matter how many levels are involved, the result of unnesting tables is a cross product of the parent and child tables. When more than one level of unnesting occurs, the inner-most child is unnested first, then the result--the cross product of the parent and the inner-most child--is then unnested from its parent, and so on to the top-level table, creating the result shown in Fig. 9C.

Unnesting all tables (cross product of all data) may not produce the results intended. For example, if multiple customer values are included in an order, such as ship-to and bill-to addresses, flattening a sales order by unnesting customer and line item tables produces rows of data that may not be useful for processing the order. This is illustrated in Fig. 10. Using the GUI, the specification of the data flow is shown in Fig. 11.

A DTD (document type definition) describes the data schema of an XML message or file. Real-time data flows read and write XML messages based on a specified DTD format. One DTD can describe multiple XML sources or targets. Batch data flows can read and write data to files based on a specified DTD format.

DTDs can be imported into the NRDM system, either directly or by importing an XML document that contains a DTD. During import, the NRDM system

converts the structure defined in the DTD into an internal nested-relational data model. Elements below the root-level that contain other elements become nested tables and elements that do not contain other elements become columns. Attributes become columns in the corresponding element's schema.

5           The NRDM system applies the following rules to convert the DTD to tables, columns, and nested tables:

- Any element that contains PCDATA only and no attributes becomes a column.
- Any element with attributes or other elements (or in mixed format) becomes a table.
- An attribute becomes a column in the table corresponding to the element it supports.
- 10 - Any occurrence of choice operators is converted to strict ordering.
- Any occurrence of optional operators is converted to strict ordering.
- Any occurrence of  $()^*$  or  $()^+$  becomes a table with an internally generated name--an implicit table.

15           After these rules have been applied, the NRDM system optimizes the format using two more rules, except where doing so would allow more than one row at the root element:

- If an implicit table contains one and only one nested table, then the implicit table can be eliminated and the nested table can be attached directly to the parent of the implicit table. For example, the SalesOrder element might be defined as follows in the DTD:

20           <!ELEMENT SalesOrder (Header, LineItems\*)>

25           When converted, the LineItems element with the zero or more operator would become an implicit table under the SalesOrder table. The LineItems element itself would be a nested table under the implicit table, as shown in Fig. 12A. Because the implicit table contains one and only one nested table, the format would be optimized to remove the implicit table, as shown in Fig. 12B.

- If a nested table contains one and only one implicit table, then the implicit table can be eliminated and its columns placed directly under the nested table. For example, the nested table LineItems might be defined as follows in the DTD:

30           <!ELEMENT LineItems (ItemNum, Quantity)\*>

35           When converted, the grouping with the zero or more operator would become an implicit table under the LineItems table. The ItemNum and Quantity elements would become columns under the implicit table, as shown in Fig. 12C. Because the

LineItems nested table contained one and only one implicit table, it would be optimized to remove the implicit table, as shown in Fig. 12D.

If the DTD contains an element that uses an ancestor element in its definition, the definition of the ancestor can be expanded for a fixed number of levels.

5 For example, given the following definition of element "A":

A: B, C  
B: E, F  
F: A, H

10 The system produces a table for the element "F" that includes an expansion of "A." In this second expansion of "A," "F" appears again, and so on until the fixed number of levels. In the final expansion of "A," the element "F" appears with only the element "H" in its definition.

#### Real-Time Sources

15 A real-time source in a real-time data flow determines the message that the real-time data flow will process. The source object represents the schema of the expected messages. Messages received are fit to the schema. Real-time data flows accept real-time source types such as Extensible Markup Language formatted (XML) messages or intermediate documents, such as IDocs published from an SAP R/3 application server.

20 The format of the XML message is specified by a document type definition (DTD). The DTD describes the schema of data contained in the message and the relationships among the elements in the data. For a message that contains information to place a sales order--order header, customer, and line items--the corresponding DTD includes the order structure and the relationship between data, as shown by the example  
25 in Fig. 13.

The following examples provide a high-level description of how real-time data flows address typical real-time scenarios. Fig. 14A shows a real-time data flow as might be used to load transactions into an ERP system, such as SAP R/3. A real-time data flow can receive a transaction from an electronic commerce application and load it to  
30 an ERP system. Using a query transform, one can include values from a data warehouse to supplement the transaction before applying it against the ERP system.

Fig. 14B shows a real-time data flow for collecting ERP data into a warehouse. Real-time data flows can receive messages from the ERP through IDocs. Each IDoc contains a transaction that the real-time data flow can load into a data

warehouse or a data mart. In this way, IDocs can be used to keep the data in a warehouse current.

Fig. 14C shows a real-time data flow for retrieving values from a cache or and ERP system. This allows for real-time data flows that use values from a data warehouse to determine whether or not to query the ERP system directly.

#### Supplementary Sources

When more data is needed than what is provided in the content of a message to complete the message processing, supplementary sources might be used. For example, processing a message that contains a sales order from an electronic commerce application that contains the customer name might require that, when the order is applied against your ERP system, more detailed customer information is needed. Inside the real-time data flow, the message is supplemented with the customer information to produce the complete document to send to the ERP system. The supplementary information may come from the ERP system itself or from a cache containing the same information cached. Examples of such data flows are shown in Figs. 15, 16A, 16B.

Tables and files (including XML files) as sources in real-time data flows can provide this supplementary information. The real-time data flow extracts data from the supplementary source as indicated by the logic defined in the real-time data flow.

Tables or files that are used as sources and have a cache option allow for the data extracted to be stored in memory until the data flow processing is complete. In real-time data flows, sources should not be cached unless the data being cached is small and is unlikely to be updated in the life of the real-time data flow.

In batch data flows, caching can improve the performance of data flow processing by reducing the number of times a set of data is read from the database or file source. In real-time data flows, however, the improvement in performance provided by caching is minimized by the likelihood that the real-time data flow reads only a small amount of data from the source for any given message. In addition, because the real-time data flow reloads cached data only when an access server shuts it down and restarts it, cached data may become stale in memory.

Tables can be sources in real-time data flows after their metadata is imported into the repository. When the real-time data flow starts, it opens a connection to the source database. This connection remains open as long as the real-time data flow is running. If a table is included in a join with a real-time source, the data set from the real-time source is included as the outer loop of the join.

R/3 tables can be sources in real-time data flows after their metadata is imported into the repository. When the real-time data flow performs a query against the R/3 table, it executes an R/3 function call to extract the data through the SAP R/3 application server. This method of extracting data from SAP R/3 is particularly well suited to extracting a small amount of specific data (on the order of 1 to 10 rows) in a real-time system, but might not work well as a substitute to using R/3 data flows to produce ABAP programs to extract large amounts of data in a batch system.

Data from XML files can be used as sources in real-time data flows, if a DTD that describes the data in the file is imported.

#### 10 Supplementing Message Data

The data included in messages from real-time sources may not map exactly to requirements for processing or storing the information. If not, steps can be defined in the real-time data flow to supplement the message information. One technique for supplementing the data in a real-time source includes these steps in a real-time data flow:

1. Include a table or file as a source. In addition to the real-time source, include the files or tables that supply the supplementary information.
2. Use a query to extract needed data from the table or file. Use the data in the real-time source to find the needed supplementary data. A join expression can be used in the query so that only the specific values required from the supplementary source are extracted.

Fig. 16A shows an example where a message includes sales order information with the ultimate goal to return order status. In this case, the business logic uses the customer number and priority rating to determine the level of status to return. The message includes only the customer name and the order number. The real-time data flow is then defined to retrieve the customer number and rating from other sources before determining the order status.

A real-time data flow might include logic to determine when responses can be generated from data in a cache and when they must be generated from data in an ERP system. One technique for constructing this logic includes the steps in the real-time data flow (illustrated in Figs. 17-20):

1. Determine the rule for when to access the cache and when to access the ERP system.
2. Compare data from the real-time source with the rule.
3. Define each path that could result from the outcome. Consider the case where the rule indicates ERP access, but the ERP system is not currently available.
4. Merge the results from each path into a single data set.
5. Route the single result to the real-time target.

This example describes a section of a real-time data flow that processes a new sales order. The section is responsible for checking the inventory available of the ordered products--it finds an answer to the question, "is there enough inventory on hand to fill this order?" The rule controlling access to the ERP system indicates that the inventory (Inv) must be more than a pre-determined value (IMargin) greater than the ordered quantity (Qty) to consider the cached inventory value acceptable. The comparison is made for each line item in the order.

Fig. 18 illustrates a branch in the data flow based on a rule. An XML source contains the entire sales order, yet the data flow compares values for line items inside the sales order. The XML target that ultimately returns a response requires a single row at the top-most level. Because this data flow needs to be able to determine inventory values for multiple line items, the structure of the output requires the inventory information to be nested. The input is already nested under the sales order; the output can use the same convention. In addition, the output needs to include some way to indicate that the inventory is or is not available.

Fig. 19 illustrates several ways to return values from the ERP. For example, a lookup function or a join on the specific table could be used in the ERP system. The example uses a join so that the processing can be performed by the ERP system rather than the NRDM system. As in the previous join, if a value might not be returned by the join, an outer join can be defined so that the line item row is not lost.

Fig. 20 illustrates a GUI used to specify transformations and a specific transformation specified with that GUI.

Fig. 21 is a block diagram of a schema converter. In the example shown, an NRDM schema is converted to a DTD schema.

### Other Uses

One of the advantages of operating a transformation engine on NRDM data structures, as described above, is that the transformation engine can operate on hierarchical data as if it were a relational table. Thus, hierarchical documents, such as XML documents can be operated on using declarative statements, such as SQL, regardless of how many levels of hierarchy are present. One method of effecting such a benefit is to nest child tables into columns of parent tables and use a transformation engine that handles NRDM data as its input and as its output. The transformation engine can be sandwiched between an importer that converts hierarchical documents into NRDM data structures and an exporter that generates hierarchical documents from NRDM data structures.

There are various ways to implement NRDM data structures. For example, conventional relational tables can be used, where a column of the parent table stores a pointer to a child table. A separate child table could exist for each row of the parent table that does not have a NULL value for that row and column, or where the child tables for each row have corresponding formats, the data representing the child tables could be implemented as subtables of one child data-holding table. Regardless of the underlying structure, the transformation engine deals with the data structures as nested tables and applies declarative statements accordingly.

Other aspects of the system described herein might find uses apart from NRDM data structures and systems. For example, requests received from applications for data processing and/or transformation might operate on nested tables, but might also operate on conventional relational tables.

The applications often provide application programming interfaces (APIs) through which other programs interact with the application. Often, the designer of a program that interacts with the application must know the interfaces and correctly specify the parameters of a particular function call. However, some applications might accept as an input NRDM data or a hierarchical document. In some cases, the application interface could be such that the semantics of the function call are in a document submitted as a parameter and then one generic interface is all that is needed to call the application.

### Example Implementation

An example of an NRDM system according to various aspects of the present invention will now be described. It should be understood that the invention is not limited to this specific example. The example system supports hierarchical data models

such as IDoc and XML and provides for a hierarchical structure to support a hierarchical data model represented as a single row that contains scalar columns and repeating group(s) of embedded rows forming nested table(s), where nesting can be arbitrarily deep and an implicit relationship is not required between embedded rows and parent (i.e., the children rows do not need to contain a key to join it back to the parent row).

The NRDM system can capture an entire business transaction in a single hierarchical structure and transform a hierarchical structure as a single entity using relation operators that can be applied at any level of the hierarchy. A hierarchical structure when applied as a single database transaction can be loaded to a set of tables belonging to a single datastore.

#### Data Model

In NRDM, the first normal form requirement that a column be a scalar is removed. In NRDM, a column can be a scalar or a relation value, which we refer to as a nested table. A scalar column definition has a name, type (including length, precision, domain info, etc.) and, at run time, contains either a value or a NULL indicator. A nested table definition has a name, schema (e.g., a list of column definitions) and, at run time, contains either one or more rows of the schema specified in the nested table definition or an empty table indicator (e.g., IEMPTY).

#### DDL Operations

AL\_NESTED\_TABLE is used below to define a nested table for DDL operations. For example, creating a view with nested table might be done by the following statements:

```
CREATE VIEW V1 (
    ORDER_ID INT,
    PROD_INFO AL_NESTED_TABLE (
        PROD_ID INT,
        QTY INT,
        VENDOR_INFO AL_NESTED_TABLE (VNDR_ID CHAR(5),
                                     VNDR_CITY CHAR(65))
    ),
    CID INT,
    CCITY CHAR(65)
);
```

Fig. 22 illustrates a data table that might result for the above statements.

#### DML Operations

Relational operations such as select, project, etc. can be used on NRDM data. Nested relations can be accessed as regular relations in the context (scope) of their parents. In other words, wherever a scalar column is used, a nested table can be used. If a parent table is used in a FROM clause, all the nested tables can be used in the SELECT

and WHERE clauses and nested subqueries as full-fledged tables. If two parent tables having a same name for a nested table are used in a relational operation, the nested tables should be qualified with the parent tables.

Nested subqueries allow for accessing and transforming data inside nested relations. Nested subqueries can transform data in nested relations, nest, unnest and join data in nested relations with the data in its parents and handle operations such as ISEMPY, AL\_NEST, AL\_NEST\_SET and AL\_UNNEST for NRDM data. The AL\_NEST operator creates partitions based on the formation of equivalence classes to generate nested tables. It operates on a row basis. AL\_NEST\_SET operator is similar to AL\_NEST but operates on a set basis. The AL\_UNNEST operator transforms a relation into one, which is less deeply nested by concatenating each tuple in the relation being unnested to the remaining attributes in the relation.

The AL\_NEST operator creates partitions based on the formation of equivalence classes to generate nested tables. Two tuples are equivalent if they have the same values for attributes, which are not being nested. AL\_NEST operates on a row basis. Nesting can be done in two ways using a user interface (such as the GUI described above). A nested table can be dragged from the input to the output of a query transform and placed at the same or deeper level, or a nested schema can be created and columns from the input can be dragged and dropped into the newly created schema.

An explicit FROM clause might be needed where two views are coming into a query transform, and columns are selected from only one the views. The generated language is to select from both the views. For nesting of two input views containing only scalar columns, selecting from the both the views at the same level might not be desired. The following example illustrates this. Given a flat view V1 as:

```
25 CREATE VIEW ORDERS (ORDER_ID INT, PROD_ID INT, QTY INT,
    CID INT, CCITY VARCHAR(65))
```

```
CREATE VIEW VENDORS (PROD_ID INT, VNDR_ID VARCHAR(5),
    VNDR_CITY VARCHAR(65))
```

30

the table of flat relations shown in Fig. 23 results. A two level nesting to include vendor information using a JOIN can be demonstrated by the following example:

```
35 CREATE VIEW V2 (ORDER_ID INT,
    PROD_INFO AL_NESTED_TABLE (PROD_ID INT,
    QTY INT,
    VENDOR_INFO
    AL_NESTED_TABLE (
    VNDR_ID CHAR(5),
    VNDR_CITY CHAR(65))
```

```

    ),
    CID,
    CCITY
5      )
  AS SELECT ORDER_ID,
    AL_NEST (CREATE VIEW PROD_INFO (PROD_ID INT, QTY INT)
      AS SELECT PROD_ID,
        QTY,
10      AL_NEST (CREATE VIEW VENDOR_INFO
        (VNDR_ID CHAR(5),
        VNDR_CITY CHAR(65)) AS
        SELECT VNDR_ID, VNDR_CITY
        FROM VENDORS
        WHERE VENDORS.PROD_ID = L1.PROD_ID
15      )
      AS VENDOR_INFO
      FROM ORDERS L1
      WHERE L1.ORDER_ID = L0.ORDER_ID          AND
        L1.CID = L0.CID                        AND
        L1.CCITY = L0.CCITY
20      )
    AS PROD_INFO,
    CID,
25    CCITY
  FROM ORDERS L0

```

The explicit FROM clause prevents the usage of the VENDORS in the outermost select. This may produce a nested table as shown in Fig. 22, except with three
 30 rows with ORDER\_ID equal to 100, two rows with ORDER\_ID equal to 200 and one row with ORDER\_ID = 300, because AL\_NEST operates on a row basis, which can produce duplicates.

The AL\_NEST operator may be used to perform nesting on a set of rows also. If there is a GROUP BY, the set formed by the GROUP BY is used. If there are
 35 aggregate functions and a GROUP BY is specified, the set formed by the GROUP BY is used. If there are aggregate functions and a GROUP BY is not specified, then the default grouping is the entire table. All nested tables in the set operated by the AL\_NEST may be merged.

#### Using AL\_NEST\_SET with an Aggregate Function

40 This operation may take in a view with nested tables and produce a single row, which has count of ORDER\_ID's and the merge of all nested tables:

```

  CREATE VIEW V2 (NUM_ORDERS INT,
    PROD_INFO AL_NESTED_TABLE (PROD_ID INT,
    QTY INT
45      )
  )
  AS SELECT COUNT(ORDER_ID),
    AL_NEST_SET (CREATE VIEW PROD_INFO (PROD_ID INT,
    QTY INT) AS

```

```

SELECT PROD_ID, QTY
  FROM PROD_INFO
)
AS PROD_INFO,

```

5

FROM V1

Such a query might produce the table shown in Fig. 24. If the nested table(s) SELECT(S) have WHERE clauses, the nested table(s) might first be merged and the filters applied to the merged table(s).

10 AL\_UNNEST

The AL\_UNNEST operator transforms a relation into one that is less deeply nested by concatenating each tuple in the relation being unnested to the remaining attributes in the relation. To unnest the vendor information from the nested table in Fig. 22, the following ATL might be defined:

```

15  CREATE VIEW V2 (ORDER_ID INT,
      PROD_INFO AL_NESTED_TABLE (PROD_ID INT,
                                  QTY INT,
                                  VNDR_ID CHAR(5)))
      AS SELECT ORDER_ID,
20      AL_NEST (CREATE VIEW PROD_INFO (PROD_ID INT, QTY INT) AS
      SELECT V1.PROD_INFO.PROD_ID,
      V1.PROD_INFO.QTY,
      AL_UNNEST (CREATE VIEW VDR_INFO
25      (VNDR_ID INT) AS
      SELECT
      V1.PROD_INFO.VENDOR_INFO.VNDR_ID
      FROM V1.PROD_INFO.VENDOR_INFO)
      FROM V1.PROD_INFO)
      AS PROD_INFO
30  FROM V1

```

WHERE clauses can be applied in the SELECT for unnesting by drilling into the nested table which would produce a query transform, specifying the condition there, as shown in the following example:

```

35  CREATE VIEW V2 (VNDR_ID CHAR(5), VNDR_CITY CHAR(65))
      AS SELECT DISTINCT AL_UNNEST (CREATE VIEW
      UNEST1(VNDR_ID CHAR(5),
      VNDR_CITY CHAR(65))
      AS SELECT
40      AL_UNNEST (CREATE VIEW
      UNEST2(VNDR_ID CHAR(5),
      VNDR_CITY
      CHAR(65))
      AS SELECT VNDR_ID, VNDR_CITY
      FROM VENDOR_INFO)
      FROM PROD_INFO
45      )
      FROM V1

```

Project

50 An example of a simple projection from one hierarchical structure to another would be:

```

CREATE VIEW V2 (
    ORDER_ID INT,
    PROD_INFO AL_NESTED_TABLE (PROD_ID INT, QTY INT),
)
5  AS SELECT ORDER_ID,
    AL_NEST (CREATE VIEW PROD_INFO (PROD_ID INT, QTY INT)
    AS SELECT V1.PROD_INFO.PROD_ID, V1.PROD_INFO.QTY
        FROM V1.PROD_INFO)
    AS PROD_INFO

```

10 FROM V1  
 The qualifier V1.PROD\_INFO in the nested relation is not really needed; the nested query could have been written using just PROD\_INFO. The result might be the table shown in Fig. 25.

#### Select

15 Filter conditions can be applied at various levels. Consider the example of view V1 (Fig. 22) that has three levels of nesting. A filter on the nested relation PROD\_INFO might be implemented as follows:

```

CREATE VIEW V3 (ORDER_ID INT,
    PROD_INFO AL_NESTED_TABLE (PROD_ID INT, QTY INT)
20 )
    AS SELECT
        ORDER_ID,
        AL_NEST (CREATE VIEW PROD_INFO (PROD_ID INT, QTY INT)
        AS SELECT V1.PROD_INFO.PROD_ID,
            V1.PROD_INFO.QTY
25         FROM V1.PROD_INFO
            WHERE V1.PROD_INFO.QTY > 50)
        AS PROD_INFO
    FROM V1
30

```

30 This may select all the rows from V1, but for the nested table PROD\_INFO, only those rows are chosen where the quantity ordered QTY is greater than 50, resulting in the table shown in Fig. 26.

#### Alternate Support For Filters In The WHERE Clause

35 For a nested table to be used in a WHERE clause sub-query, support within a WHERE clause should be available. If such support is not available, it can be overcome by using two stages and the ISEMPY operator for nested tables. Nested tables can be used in a WHERE clause only with the ISEMPY operator. The following example illustrates the use, selecting all the rows from V1 that have ORDER\_ID greater

40 than 100 and that have at least one product with a quantity ordered greater than 50.

```

CREATE VIEW V3 (ORDER_ID INT,
    PROD_INFO AL_NESTED_TABLE (PROD_ID INT, QTY INT),
    TEMP_PROD_INFO AL_NESTED_TABLE (PROD_ID INT, QTY
45 INT)
)
    AS SELECT
        ORDER_ID,

```

```

AL_NEST(CREATE VIEW PROD_INFO(PROD_ID INT, QTY INT)
  AS SELECT V1.PROD_INFO.PROD_ID,
    V1.PROD_INFO.QTY
  FROM V1.PROD_INFO
)
AS PROD_INFO,
AL_NEST(CREATE VIEW PROD_INFO(PROD_ID INT, QTY INT)
  AS SELECT V1.PROD_INFO.PROD_ID,
    V1.PROD_INFO.QTY
  FROM V1.PROD_INFO
  WHERE V1.PROD_INFO.QTY > 50)
AS TEMP_PROD_INFO

FROM V1 WHERE V1.ORDER_ID > 100

CREATE VIEW V4 (ORDER_ID INT,
  PROD_INFO AL_NESTED_TABLE(PROD_ID INT, QTY INT)
)
AS SELECT
  ORDER_ID,
  AL_NEST(CREATE VIEW PROD_INFO(PROD_ID INT, QTY INT)
    AS SELECT V1.PROD_INFO.PROD_ID,
      V1.PROD_INFO.QTY
    FROM V1.PROD_INFO
  )
  AS PROD_INFO

FROM V3 WHERE !ISEMPTY(TEMP_PROD_INFO)

```

Join

Nested relations can be joined with any other relations. An example is given below:

```

CREATE VIEW ORDERS (ORDERID INT, PRODUCTS
  AL_NESTED_TABLE (PRODID INT, PRODNAME VARCHAR (10)));

CREATE VIEW VENDORS (PRODID INT, VENDORID INT,
  VENDORNAME VARCHAR (10));

CREATE VIEW ORDERS_WITH_VENDORS (ORDERID INT,
  PRODUCTS AL_NESTED_TABLE (PRODID INT,
    PRODNAME VARCHAR (10),
    VENDORID INT)
  AS
  SELECT ORDERID,
    AL_NEST (CREATE VIEW PRODUCTS (PRODID INT,
      PRODNAME VARCHAR (10),
      VENDORID INT)
    AS SELECT PRODID, PRODNAME, VENDORID
      FROM PRODUCTS, VENDORS
      WHERE PRODUCTS.PRODID = VENDORS.PRODID)
  AS PRODUCTS
  FROM ORDERS GROUP BY ORDERID

```

### Nested Table Transform

A system transform is available that takes in a flat view and produces a singleton that has a N integer scalar column with a value 1, and a nested table containing the input view.

#### 5 Tables as Parameters

Tables can be used as parameters for imported functions. Given a function get\_orders with an input parameter customer\_id and an output parameter orders:

```

10 CREATE FUNCTION get_orders (cust_id int,
                                orders AL_NESTED_TABLE (order_id int, ...)
                                OUTPUT,
                                cust_info AL_NESTED_TABLE (cust_name, ...)
                                OUTPUT);

```

Get orders for each customer by calling the orders function:

```

15 CREATE VIEW customer_orders (customer_id int,
                                orders AL_NESTED_TABLE (order_id
                                                            int, ...))
    AS SELECT customer_id,
              AL_NEST (get_orders (customer_id)::orders)
20              AS orders
    FROM customers;

```

if the function has multiple tables as outputs, and all or some of them are required, then the function has to be invoked multiple times: once for each output.

```

25 CREATE VIEW customer_orders (customer_id int,
                                cust_info AL_NESTED_TABLE (cust_name, ...),
                                orders AL_NESTED_TABLE (order_id
                                                            int, ...))
    AS SELECT customer_id,
              AL_NEST (get_orders (customer_id)::cust_info) AS
30              cust_info
              AL_NEST (get_orders (customer_id)::orders) AS orders
    FROM customers;

```

As an optimization, the system could invoke the function only once and use those results for different instances within the query transform. For mapping a function returning

35 table, a user would create a nested table column and map the nested table column to the function returning a table. The schema of the nested table may then be identical to the schema returned by the function. This is a concept of a "generated table". The schema definition of generated table cannot be modified, and it should disappear when the function is removed from the mapping. It should be represented differently in the UI so

40 that a user can distinguish between a generated table and a non-generated table.

### Hierarchical File Reader

A hierarchical file reader reads data generated by data flows that have functions that return tables. There are two main alternatives: model the file reader as an

XML file reader or model the file reader using a proprietary format to represent hierarchical data.

#### Effect of NRDM on System Transforms

System transforms such as Table\_Comparison, Hierarchy\_Flattening, etc.  
5 accept only rows with scalar columns.

Table Comparison: The output schema of the table comparison transform is a generated schema and is same as the schema of the table being compared against. This transform may silently ignore columns that are nested tables.

10 History Preserving: The output schema of the history preserving transform is same as the input schema, and this transform may preserve history only scalar columns and may act as pass through for columns that are nested tables.

Effective Date: The transform may act as pass through for columns that are nested tables.

15 Key Generation: The output schema of the key generation transform is same as the input schema, and this transform may act as pass through for columns that are nested tables.

Map Operation: The output schema of the map operation transform is same as the input schema, and this transform may not allow operations to be mapped for columns as nested tables and may act as pass through for them.

20 Hierarchy Flattening: Columns as nested tables cannot be a parent or child column of a hierarchy, but they can be dragged and dropped attribute columns and thus can appear in the output schema.

Pivot: The output schema of the hierarchy flattening transform is a generated schema and columns, as nested tables may be ignored.

#### 25 A Case Study

A case study of a Sales Order IDoc using NRDM was performed. The IDoc was captured in a NRDM and perform transformations, to arrive at the same result as if the NRDM was not used, but with simplified specification of the transformations.

30 An IDoc is divided into a control record, data records and a status record. Each control record and status record has numerous fields. For our purpose of validating the NRDM, we treated control records and status records as single varchar columns. The ATL to represent a Sales Order (some of the columns associated with nested tables might be omitted in the listing) is:

```

CREATE VIEW V1 (
  CONTROL_RECORD VARCHAR (100),
  STATUS_RECORD VARCHAR (100),
  E2CMCCO AL_NESTED_TABLE (
5      ZEITP VARCHAR (2), ... ,
      E2CVBUK AL_NESTED_TABLE (
          SUPKZ VARCHAR (1), ... ,
          E2CVBAK AL_NESTED_TABLE (
10              SUPKZ VARCHAR (1), ... ,
              E2CVBK0 AL_NESTED_TABLE (
                  SUPKZ VARCHAR (1),
              ),
              E2CVBP0 AL_NESTED_TABLE (
                  SUPKZ VARCHAR
15                      (1),
              ),
              E2CVBAP AL_NESTED_TABLE (
                  SUPKZ VARCHAR (1),
                  E2CVBA2
20      AL_NESTED_TABLE (
          SUPKZ
          VARCHAR(1),
          ),
          E2CVBUP
25      AL_NESTED_TABLE (
          SUPKZ
          VARCHAR(1),
          ),
          E2CVBPF
30      AL_NESTED_TABLE (
          SUPKZ
          VARCHAR (1)
          ),
          E2CVBKD
35      AL_NESTED_TABLE (
          SUPKZ
          VARCHAR (1),
          ),
          E2CKONV
40      AL_NESTED_TABLE (
          SUPKZ
          VARCHAR (1),
          ),
          E2CVBPA
45      AL_NESTED_TABLE (
          SUPKZ
          VARCHAR (1),
          ),
          E2CVBFA
50      AL_NESTED_TABLE (
          SUPKZ
          VARCHAR (1),
          ),
          E2CFPLT
55      AL_NESTED_TABLE (
          SUPKZ
          VARCHAR (1),
          ),
          E2CVBEP
60      AL_NESTED_TABLE (

```

```

                                SUPKZ
                                VARCHAR (1),
                                ),
                                ), # E2CVBAP
5                                ), # E2CVBAK
                                ), # E2CVBUK
                                ) # E2CMCC0
                                ) # V1

```

10 The ATL corresponding to the population of the sales order fact table from the above view may be (with some columns omitted for illustration purposes):

```

CREATE VIEW V2 ( SO_NUM,          # VBAK.VBELN
                 SOLD_TO,        # VBAK.KUNNR
15                 LINE_ITEM_ID,  # VBAP.POSNR
                 CREATE_DATE,    # VBAP.ERDAT
                 SHIP_TO,        # VBPA.KUNNR
                 DELIVERY_STATUS # VBUP.LFGSA
                 ),
20 AS SELECT AL_UNNEST
      (SELECT AL_UNNEST
        (SELECT AL_UNNEST
          (SELECT VBELN, KUNNR,
25              AL_UNNEST (SELECT POSNR, ERDAT,
                          AL_UNNEST (SELECT KUNNR FROM
E2CVBPA
                                   WHERE PARVW =
'WE'),
                                   AL_UNNEST (SELECT LFGSA FROM
30 E2CVBUP)
                                   FROM E2CVBAP
                                   )
                                   FROM V1.E2CMCC0.E2CVBUK.E2CVBAK
                                   )
35              FROM V1.E2CMCC0.E2CVBUK
              )
              FROM V1.E2CMCC0
              )
      FROM V1
40

```

WHAT IS CLAIMED IS:

- 1                    1. An apparatus for processing data representable in a hierarchical form,  
2     the apparatus comprising:  
3         an importer having inputs to receive a schema and a structured document from a data  
4             source, wherein the importer outputs a first nested relational data model  
5             (NRDM) data structure representing the structured document according to the  
6             received schema;  
7         an transformation engine that is capable of transforming the first NRDM data  
8             structure output by the importer into a second NRDM data structure according to  
9             a declarative specification of a transform; and  
10        an exporter having an input to receive the second NRDM data structure, wherein the  
11            exporter outputs a transformed hierarchical document in a data structure other  
12            than an NRDM data structure in a form suitable for a data target.
- 13                   2. The apparatus of claim 1, further comprising means for converting  
14     relational data to an NRDM data structure by vertically partitioning a relation and nesting  
15     parts of the relational data as a nested table.
- 16                   3. The apparatus of claim 1, further comprising means for converting  
17     nested relational data to relational data by unnesting the nested tables using a  
18     cross-product between a parent row and a child subtable.
- 19                   4. The apparatus of claim 1, further comprising means for performing a  
20     grouping operation on a nested table that generates a resulting nested table containing a  
21     union of all the nested tables grouped by the operation.
- 22                   5. The apparatus of claim 1, further comprising means for performing  
23     multi-step transformations, wherein an input to a transformation is results of a previous  
24     transformation, a data source, or both.
- 25                   6. The apparatus of claim 1, wherein the transformation engine operates on  
26     rules that are applied to data independent of data format.
- 27                   7. The apparatus of claim 1, wherein the exported is adapted to output one  
28     or more of an XML file, a relational table or a flat file.

29           8. A metadata mapper comprising:  
30           an input for receiving a document description for hierarchical documents; and  
31           an output for outputting an NRDM data structure representing the document  
32           description.

33           9. An apparatus for transforming data representable in a hierarchical form,  
34   the apparatus comprising:  
35           an importer having inputs to receive a schema and a structured document from a data  
36           source, from a data transformer, or from both, wherein the importer outputs a  
37           first nested relational data model (NRDM) data structure representing the  
38           structured document according to the received schema;  
39           an transformation engine that is capable of transforming the first NRDM data  
40           structure output by the importer into a second NRDM data structure according to  
41           a declarative specification of a transform; and  
42           an exporter having an input to receive the second NRDM data structure, wherein the  
43           exporter outputs a transformed hierarchical document in a data structure other  
44           than an NRDM data structure in a form suitable for a data target.

45           10. A method for providing data to an application through a data platform  
46   in a computer system in response to request from the application, the method comprising:  
47           accepting declarative rules for accessing the data from data sources and declarative  
48           rules for transforming the data into a format requested by the application;  
49           mapping relational and non-relational data sources to an NRDM data structure;  
50           interpreting a request;  
51           retrieving data from the data sources;  
52           transforming the data according to the declarative rules; and  
53           returning the transformed data to the application.

54           11. The method of claim 10, wherein requests are processed as messages  
55   and request messages contain sufficient information to drive data extraction into a  
56   data-oriented interface.

57           12. The method of claim 10, wherein the requests are application  
58   programming interface function calls.

59                   13. A method for updating a plurality of data targets from a message,  
60 comprising:  
61       making an update request through a data-oriented interface;  
62       specifying declarative rules for updating the data targets;  
63       importing metadata that maps relational and non-relational data targets to NRDM  
64       data structures;  
65       interpreting incoming update requests;  
66       transforming the data according to the declarative rules; and  
67       updating the data targets.

68                   14. The method of claim 13, further comprising:  
69       making an update request using an application; and  
70       causing one of a response to be sent to the application, an update of data, or both.

71                   15. The method of claim 13, further comprising a step of combining the  
72       update request with other data before updating the data targets.

73                   16. A method of providing input to an application expecting one or more  
74       tables as parameters to an input message, the method comprising:  
75       mapping data in a NRDM data structure to function parameters; and  
76       making a function calls to the application using the NRDM mapped data structure.

Order data set

OrderNo	CustID	ShipTo1	ShipTo2	LineItems
9999	1001	123 State St.	Town, CA	

Item	Qty	ItemPrice
001	2	10
002	4	5

FIG. 1

Order header data set

OrderNo	CustID	ShipTo1	ShipTo2
9999	1001	123 State St.	Town, CA

Line-item data set

OrderNo	Item	Qty	ItemPrice
9999	001	2	10
9999	002	4	5

WHERE  
Header.OrderNo = LineItem.OrderNo

Order data set

OrderNo	CustID	ShipTo1	ShipTo2	Item	Qty	ItemPrice
9999	1001	123 State St.	Town, CA	001	2	10
9999	1001	123 State St.	Town, CA	002	4	5

FIG. 2 (PRIOR ART)

FIG. 3 (PRIOR ART)

Order data set

OrderNo	CustID	ShipTo1	ShipTo2	LineItems
9999	1001	123 State St.	Town, CA	

Item	ItemQty	ItemPrice
001	2	
002	4	

Qty	Sell Price
1	25
20	23

Qty	SellPrice
1	200
10	190

FIG. 4

Order data set

OrderNo	LineItems	CustInfo
9999		

Item	ItemQty	ItemPrice
001	2	
002	4	

Qty	Sell Price
1	25
20	23

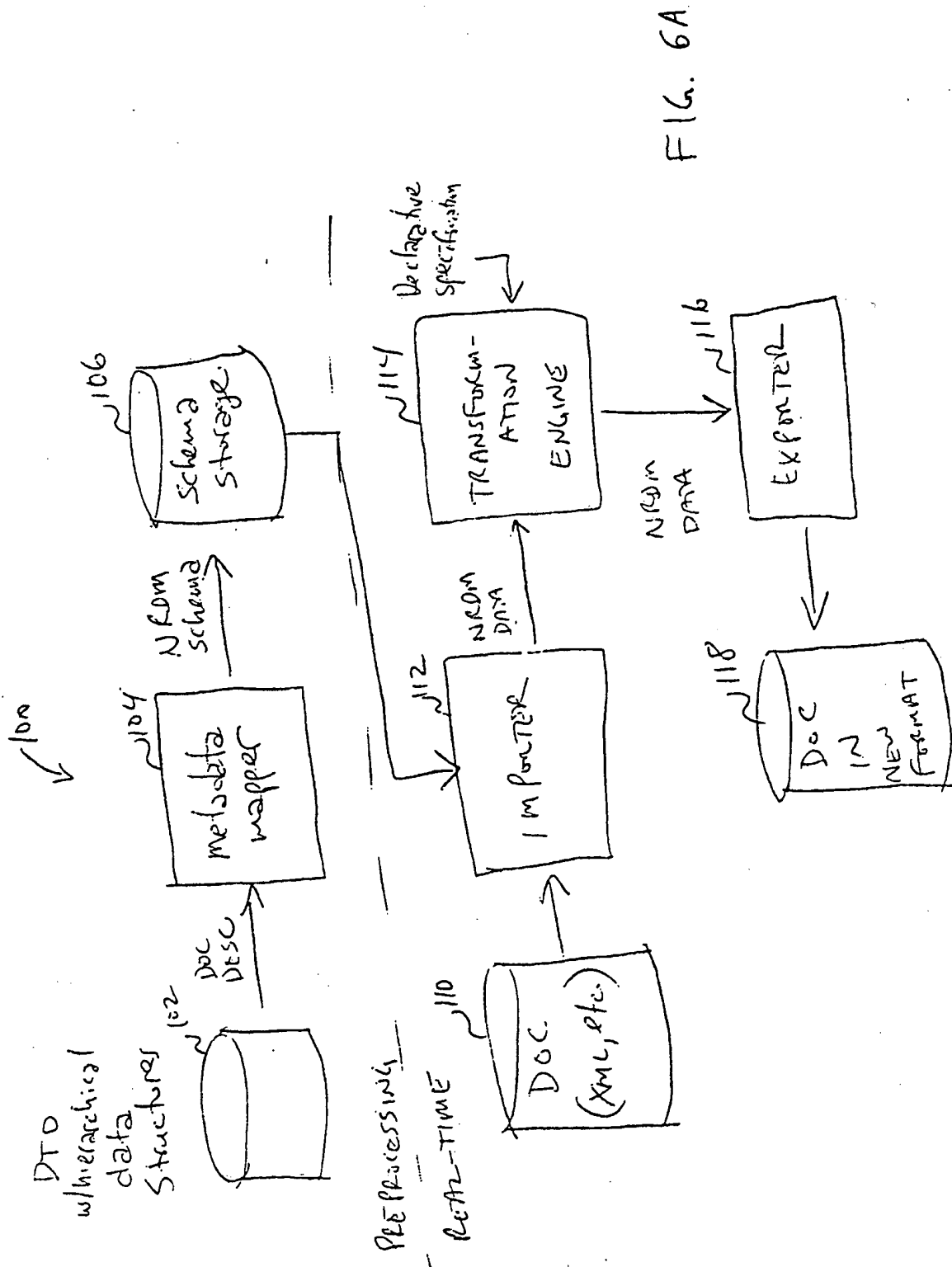
Qty	SellPrice
1	200
10	190

Type	CustID	Contacts
Ship	1001	
Bill	7777	

Name	Phone
Alvarez	555-1234
Tanaka	555-6666

Name	Phone
Samp	333-1234

FIG. 5



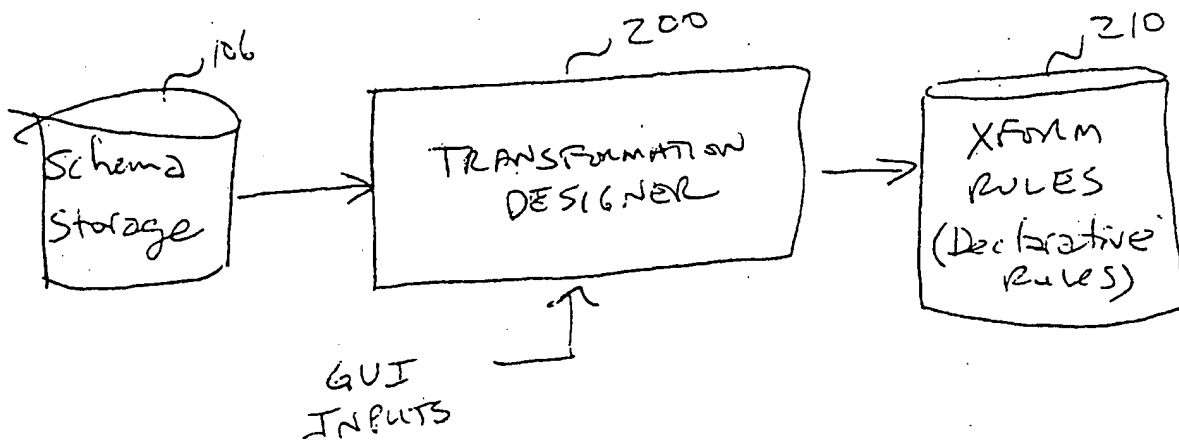


FIG. 6B

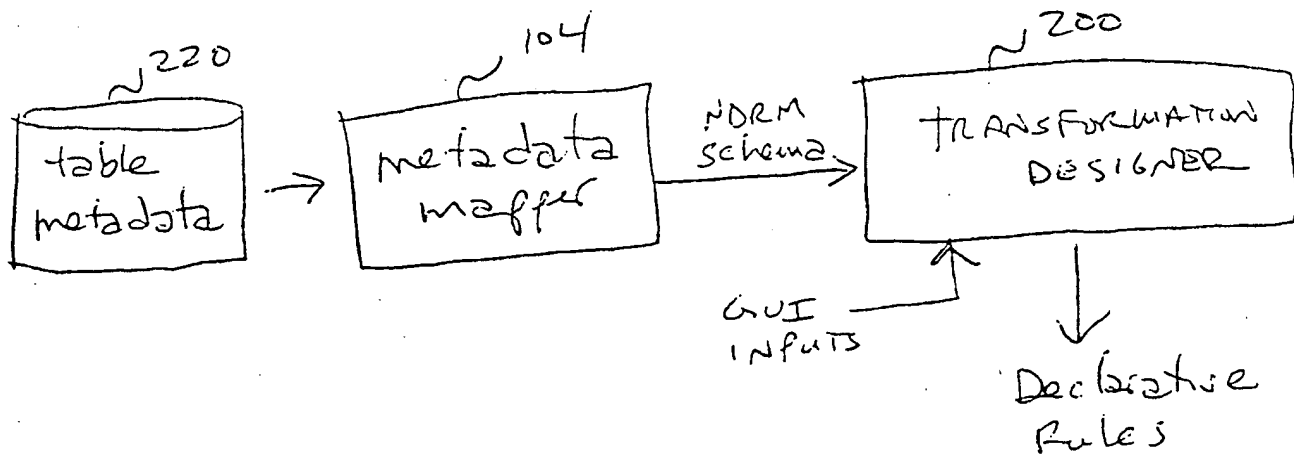


FIG. 6C

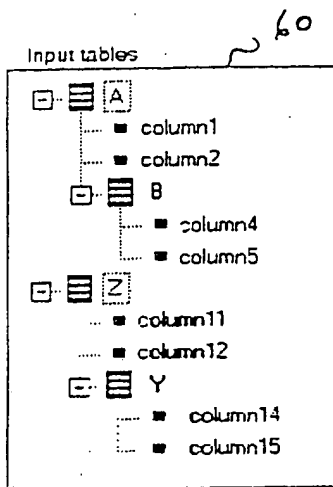


FIG. 7A

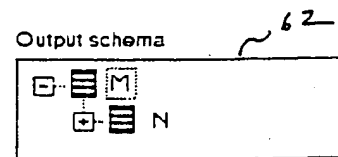


FIG. 7B

Order data set

OrderID	CustID	State	LineItems
9998	1000	CA	
9999	1001	CA	
9777	1202	TX	

Item	Qty	ItemPrice
001	2	10
002	4	5

Item	Qty	ItemPrice
001	7	23
002	7	10

Item	Qty	ItemPrice
001	9	99
002	1	2

FIG. 8

Order data set with Group By State

State	LineItems
CA	
TX	

Item	Qty	ItemPrice
001	2	10
002	4	5

Item	Qty	ItemPrice
001	7	23
002	7	10

Item	Qty	ItemPrice
001	9	99
002	1	2

Nested data set

OrderID	CustID	LineItems
9999	1001	

Item	ItemQty
001	2
002	4

Header table

OrderID	CustID	Item	ItemQty
9999	1001	001	2
9999	1001	002	4

FIG. 9A

Nested data set

OrderID	CustID	LineItems
9999	1001	

Item	ItemQty
001	2
002	4

Header table

OrderID	CustID
9999	1001

Line-item table

OrderID	Item	ItemQty
9999	001	2
9999	002	4

FIG. 9B

Order data set

OrderID	CustID	LineItems
9999	1001	

Item	ItemQty	ItemPrice
001	2	
002	4	

Qty	Sell Price
1	25
20	23

Qty	Sell Price
1	200
10	190

Unnested data set

OrderID	CustID	Item	ItemQty	Qty	SellPrice
9999	1001	001	2	1	200
9999	1001	001	2	10	190
9999	1001	002	4	1	25
9999	1001	002	4	20	23

FIG. 9C

Nested data set

OrderID	LineItems	CustInfo
9999		

Item	ItemQty
001	2
002	4

Type	CustID
Ship	1001
Bill	7777

Unnested data set

OrderID	Item	ItemQty	Type	CustID
9999	001	2	Ship	1001
9999	002	4	Ship	1001
9999	001	2	Bill	7777
9999	002	4	Bill	7777

FIG. 10

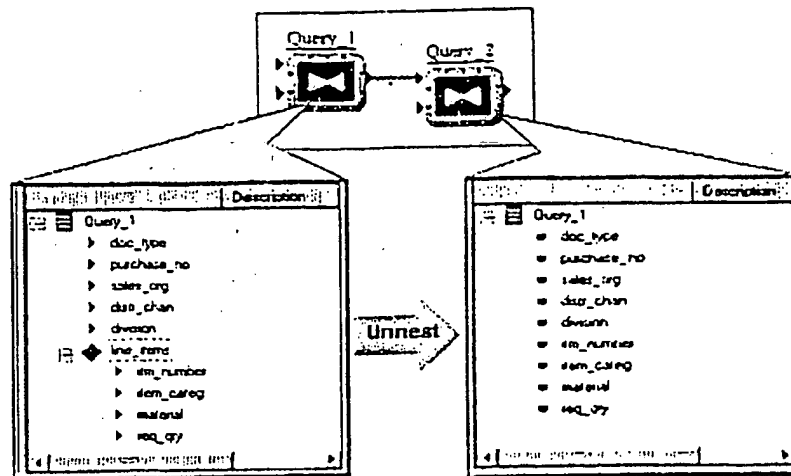


FIG. 11

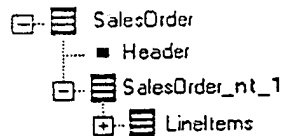


FIG. 12A

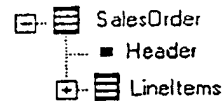


FIG. 12B

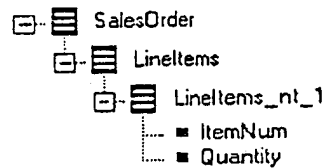


FIG. 12C

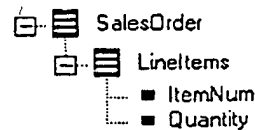


FIG. 12D

Message with data

OrderNo	CustID	ShipTo1	ShipTo2	Lineltems
9999	1001	123 State St.	Town, CA	

Item	ItemQty	ItemPrice
001	2	10
002	4	5

Each column in the message corresponds to an ELEMENT definition in the DTD.

Corresponding DTD Definition

```
<?xml encoding="UTF-8"?>
<!ELEMENT Order (OrderNo, CustID, ShipTo1, ShipTo2, Lineltems+)>
<!ELEMENT OrderNo (#PCDATA)>
<!ELEMENT CustID (#PCDATA)>
<!ELEMENT ShipTo1 (#PCDATA)>
<!ELEMENT ShipTo2 (#PCDATA)>
<!ELEMENT Lineltems (Item, ItemQty, ItemPrice)>
<!ELEMENT Item (#PCDATA)>
<!ELEMENT ItemQty (#PCDATA)>
<!ELEMENT ItemPrice (#PCDATA)>
```

FIG. 13

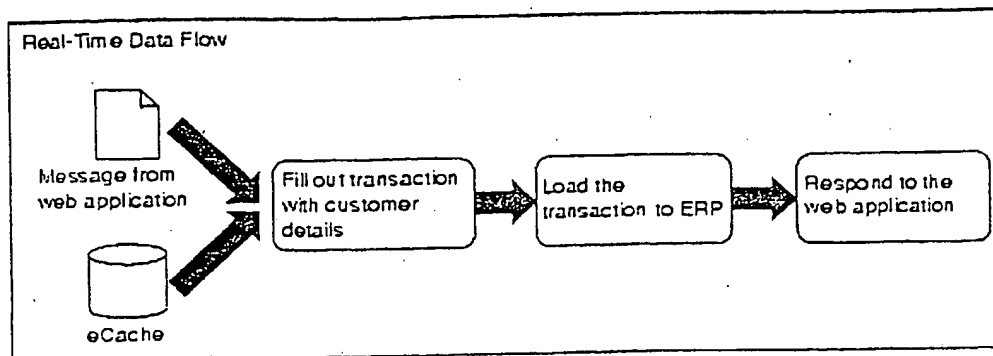


FIG. 14A

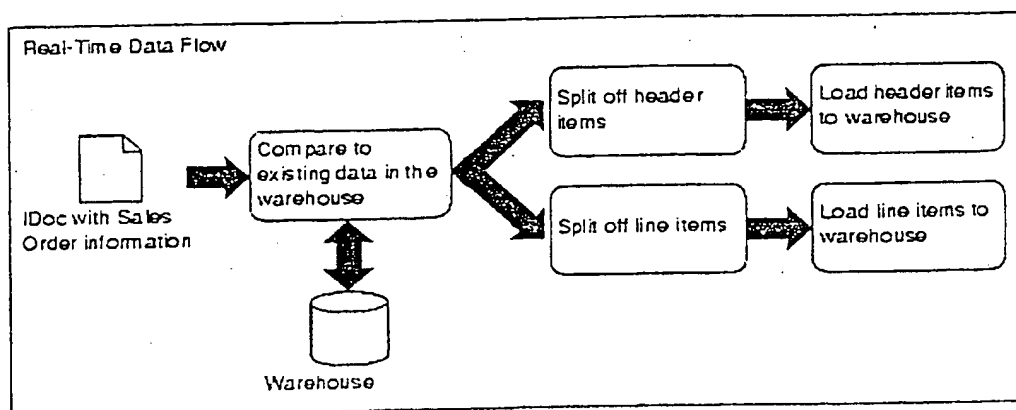


FIG. 14B

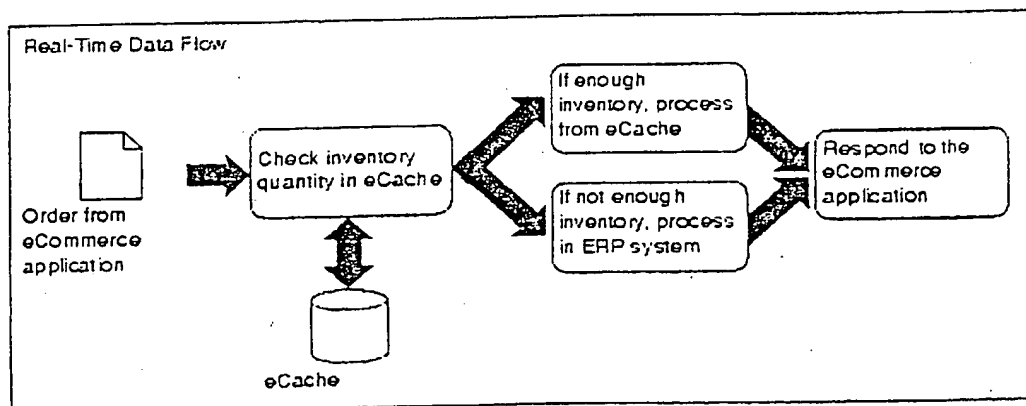
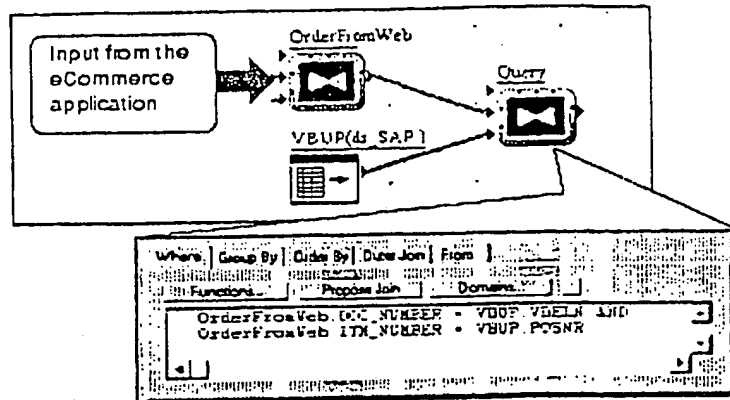


FIG. 14C



The WHERE clause joins the two inputs, resulting in output for only the sales document and line items included in the input from the eCommerce application.

FIG. 15

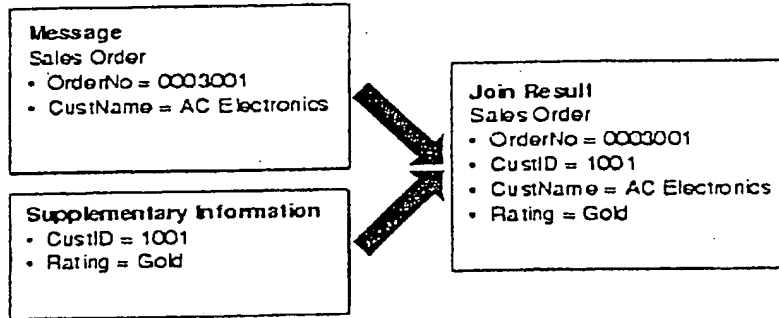


FIG. 16A

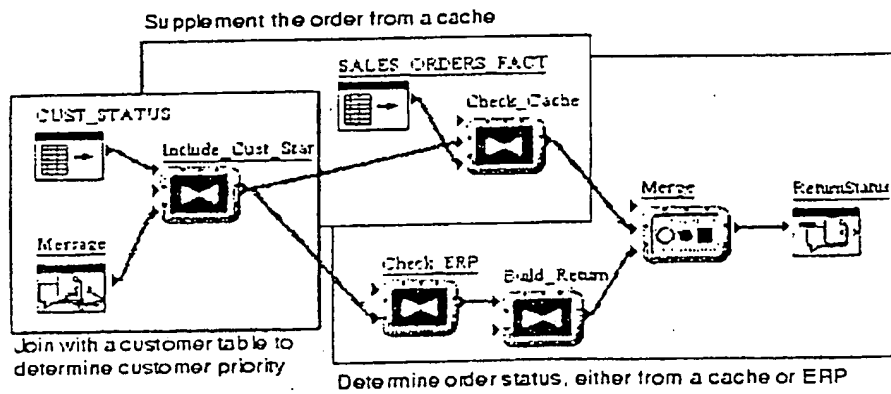


FIG. 16B

Incoming sales order

OrderID	CustID	LineItems
9999	1001	

Item	Material	Qty
001	7333	300
002	2288	1400

Inventory data in cache

Material	Inv	IMargin
7333	600	100
2288	1500	200

The quantity of items in the sales order is compared to inventory values in the cache.

FIG. 17

Incoming sales order

OrderID	CustID	LineItems
9999	1001	

Item	Material	Qty
001	7333	300
002	2288	1400

Outgoing sales order

OrderID	CustID	LineItems
9999	1001	

Item	Material	Qty	Inv
001	7333	300	
002	2288	1400	1300

FIG. 18

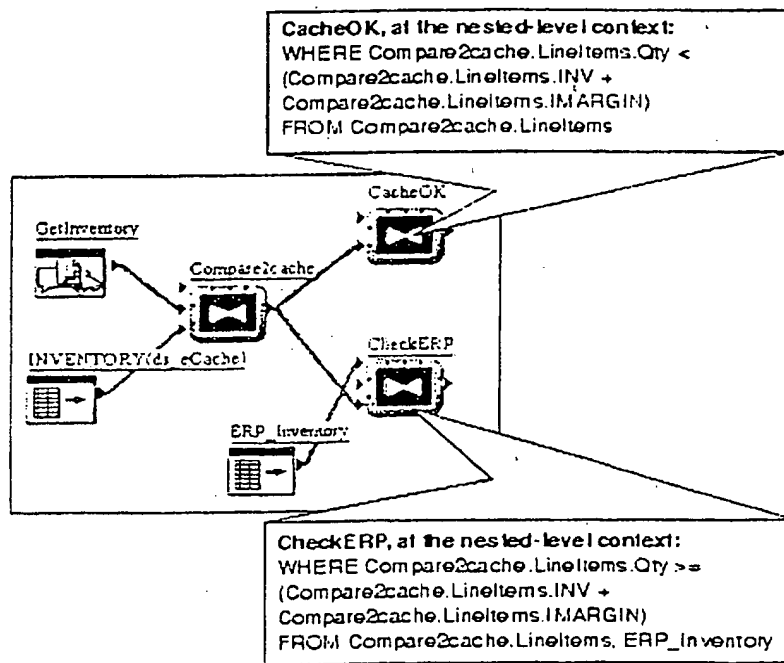


FIG. 19

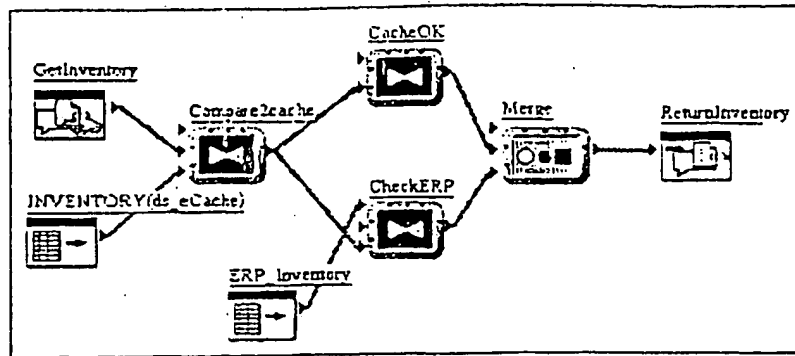
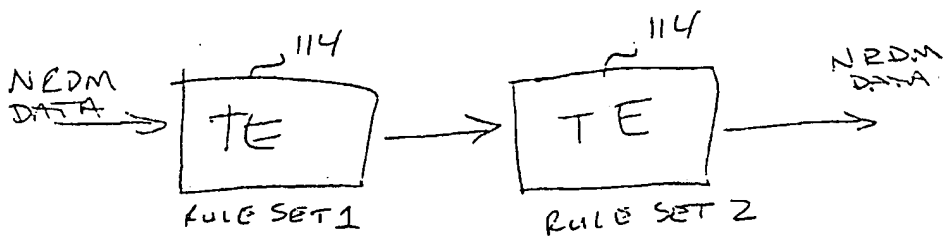


FIG. 20



FIG. 21A



ORDER_ID	PROD_INFO				CID	CCITY
	PROD_ID	QTY	VENDOR_INFO			
			VNDR_ID	VNDR_CITY		
100	101	50	10SF	SanFran	444	SanFran
			20BK	Berkley		
	201	100	10SF	SanFran		
			30SJ	SanJose		
	301	100	30SC	SantaClara		
200	201	50	10SF	SanFran	555	Berkley
			30SJ	SanJose		
	301	100	30SC	SantaClara		
			20BK	Berkley		
300	401	50	35WC	WCreek	666	Dallas

FIG. 22

## VENDORS AND ORDERS TABLES

ORDER_ID	PROD_ID	QTY	CID	CCITY
100	101	50	444	SanFran
100	201	100	444	SanFran
100	301	100	444	SanFran
200	201	50	555	Berkley
200	301	100	555	Berkley
300	401	50	666	Dallas

PROD_ID	VNDR_ID	VNDR_CITY
101	10SF	SanFran
101	20BK	Berkley
201	10SF	SanFran
201	30SJ	SanJose
301	20BK	Berkley
301	10SF	SanFran
401	35WC	WCreek

FIG. 23

NUM_ORDERS	PROD_INFO	
	PROD_ID	QTY
3	101	50
	201	100
	301	100
	201	50
	301	100
	401	50

FIG. 24

ORDER_ID	PROD_INFO	
	PROD_ID	QTY
100	101	50
	201	100
	301	100
200	201	50
	301	100
300	401	50

FIG. 25

ORDER_ID	PROD_INFO	
	PROD_ID	QTY
100	201	100
	301	100
200	301	100
300		

FIG. 26

# INTERNATIONAL SEARCH REPORT

International application No.

PCT/US01/04698

## A. CLASSIFICATION OF SUBJECT MATTER

IPC(7) : G06F 17/00  
US CL : 707/101

According to International Patent Classification (IPC) or to both national classification and IPC

## B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)  
U.S. : 707/100-104

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)  
EAST, ACM, IEL

## C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 5,937,409 A (WETHERBEE) 10 August 1999 (10.08.1999), ALL.	1-16
A	US 5,970,490 A (MORGENSTERN) 19 October 1999 (19.10.1999), ALL.	1-16

☐ Further documents are listed in the continuation of Box C.

☐ See patent family annex.

\* Special categories of cited documents:

- \*A\* document defining the general state of the art which is not considered to be of particular relevance
- \*E\* earlier application or patent published on or after the international filing date
- \*L\* document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
- \*O\* document referring to an oral disclosure, use, exhibition or other means
- \*P\* document published prior to the international filing date but later than the priority date claimed

- \*T\* later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
- \*X\* documents of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
- \*Y\* documents of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
- \*G\* document member of the same patent family

Date of the actual completion of the international search  
27 March 2001 (27.03.2001)

Date of mailing of the international search report

27 APR 2001  
27 APR 2001

Name and mailing address of the ISA/US  
Commissioner of Patents and Trademarks  
Box PCT  
Washington, D.C. 20231  
Facsimile No. (703)305-3230

Authorized officer

Thomas Black

Telephone No. 305-9000

Peggy Hamed



CORRECTED VERSION

(19) World Intellectual Property Organization  
International Bureau



(43) International Publication Date  
16 August 2001 (16.08.2001)

PCT

(10) International Publication Number  
WO 01/059602 A1

- (51) International Patent Classification?: G06F 17/00
- (21) International Application Number: PCT/US01/04698
- (22) International Filing Date: 12 February 2001 (12.02.2001)
- (25) Filing Language: English
- (26) Publication Language: English
- (30) Priority Data:  
60/182,047 11 February 2000 (11.02.2000) US
- (71) Applicant (for all designated States except US): ACTA TECHNOLOGIES, INC. [US/US]; 1667 Plymouth Street, Mountain View, CA 94043 (US).
- (72) Inventors; and
- (75) Inventors/Applicants (for US only): GORELIK, Alexander [US/US]; 44153 Boitano Drive, Fremont, CA 94539 (US). CHAWLA, Sachinder, Singh [US/US]; 1963 Pine Street, San Francisco, CA 94109 (US). SYED, Awez,

Imran [US/US]; 10810 Brewer House Road, Rockville, MD 20852 (US). BURDA, Leon [US/US]; 22206 Quinterno Court, Cupertino, CA 95014 (US). YEE, Mon, For [US/US]; 255 Yerba Buena Avenue, San Francisco, CA 94127 (US). GANTIMAHAPATRUNI, Sridhar [US/US]; 361 Sunset Avenue, Sunnyvale, CA 94086 (US).

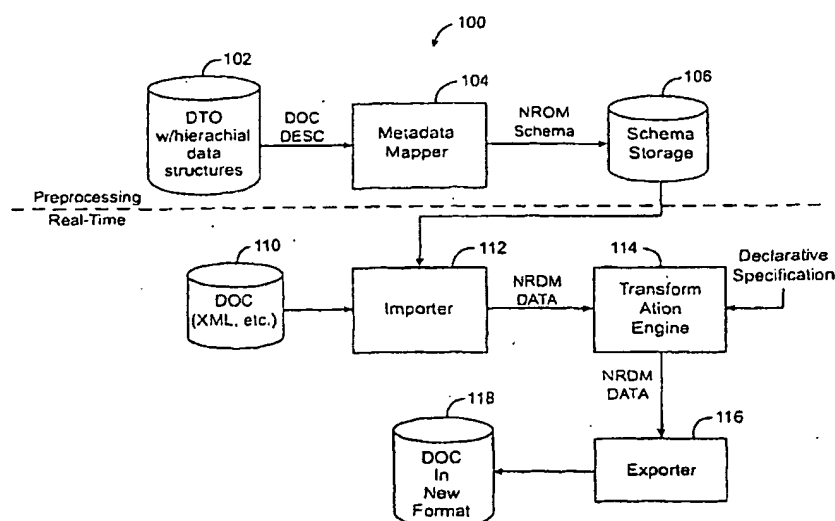
(74) Agents: ALBERT, Philip, H. et al.; Townsend and Townsend and Crew LLP, Two Embarcadero Center, Eighth Floor, San Francisco, CA 94111-3834 (US).

(81) Designated States (national): AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW.

(84) Designated States (regional): ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian

[Continued on next page]

(54) Title: NESTED RELATIONAL DATA MODEL



(57) Abstract: A data base system (100) handles nested relational data model (NRDM) data. A metadata mapper (104) converts DTD w/hierarchical structures (102) to NRDM schema that are then stored in schema storage (106). A document (110) is passed to an importer (112), then to a transformation engine (114) and an exporter (116) to result in a document in a new format (118). Because the transformation engine (114) operates on NRDM structures, the transformations can be expressed as a declarative specification, thus simplifying the process of transforming complex data. Exporter (116) exports the data in a suitable form such as XML documents, relational tables or flat files.

WO 01/059602 A1



patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:

— with international search report

(48) Date of publication of this corrected version:

17 October 2002

**(15) Information about Correction:**

see PCT Gazette No. 42/2002 of 17 October 2002, Section II

*For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.*

## NESTED RELATIONAL DATA MODEL

### COPYRIGHT NOTICE

A portion of the disclosure of this patent document contains material that is subject to copyright protection. The copyright owner has no objection to the facsimile reproduction by anyone of the patent document or the patent disclosure as it appears in the Patent and Trademark Office patent file or records, but otherwise reserves all copyright rights whatsoever.

### FIELD OF THE INVENTION

The present invention relates to information management in general and more particularly to methods for using Nested Relational Data Models (NRDMs) to manage information.

### BACKGROUND OF THE INVENTION

Information is commonly managed in units of documents. For example, sales, distribution and manufacturing information might be contained within documents such as sales invoices or orders. Increasingly, documents pass between parties in electronic form, in a process generally referred to as EDI (Electronic Data Interchange). In electronic form, the documents are not limited to the text and images shown on the printed page, but can include formatting and "metadata" (data about the data). One example of a format for an electronic document that contains metadata is the Extended Markup Language (XML).

Several products on the market allow mapping of XML documents to SQL tables or vice versa and several products on the market allow mapping of EDI documents to relational tables or vice versa, but these products typically require procedural specifications of how to perform the conversion, such as programming code. Traditional Relational Database Management Systems (RDMS's) such as described by Date or Ullman or implemented by Oracle, IBM, Microsoft and others as well as distributed databases as described in Ceri or U.S. Patent Nos. 5,884,310 and 5,596,744, implement declarative transformations of relational data.

A class of systems called intelligent gateways (such as Sybase's OmniServer system) allow declarative rules to be transparently applied to heterogeneous relational databases. Another class of systems called Replication Servers (such as

described by U.S. Patent No. 5,737,601 or implemented as Sybase's Replication Server, Oracle's Replication Server, or the like) can provide homogeneous or heterogeneous data replication.

Additional class of systems called the ETL (Extraction, Transformation, Loading) systems such as Microsoft DTS, Informatica PowerMart and D2K Tapestry provide extraction, transformation and loading of heterogeneous data between relational database systems. Some of these products support converting hierarchical files into a relational form by "flattening" the hierarchical files, making multiple passes through a hierarchical file and, at each pass, pulling out different parts of the hierarchy.

Yet another class of systems that address mapping of relational data to a programming object, as exemplified by U.S. Patent Nos. 6,175,837, 6,163,781, 6,134,559, 5,907,846, 5,873,093, 5,832,498, or products from Persistence, Bea and others. This class of tools maps persistently stored relational data to an object-oriented memory representation as well as mapping the data from an object-oriented memory representation to a set of persistent relational tables.

Another class of prior art exists that provides object-oriented access to non-relational databases, as described in U.S. Patent Nos. 5,799,313, 5,778,379, and 5,542,078. This class of systems addresses the mapping of data from hierarchical databases such as IMS, object oriented databases and relational databases to an object-oriented programming object or database.

Considerable research has been done on Nested Relational Data Models as described in \_\_\_, "Lecture Notes in Computer Science Volume 595: M. Levene – The Nested Universal Relation Database Model" and \_\_\_, "Lecture Notes in Computer Science Volume 361: S. Abiteboul et al. – Nested Relations and Complex Objects in Databases". That research focused mainly on defining the data model and specific operations on it.

It is known to graphically map disparate schemas to each other. See, for example, U.S. Patent Nos. 5,850,631 and 5,806,066. It is also known to map data between different structures. See for example, U.S. Patent Nos. 5,627,972 and 5,119,465.

### SUMMARY OF THE INVENTION

In one embodiment of data processing system according to the present invention, hierarchical documents or hierarchical messages are mapped to a Nested Relational Data Model to allow for transformation and manipulation using declarative

statements. The resulting nested data can be converted to a relational format and mapped to multiple relational tables, and/or converted from a nested relational format to an external hierarchical format, such as XML.

The system can specify and execute declarative rules to extract, transform, integrate, load and update hierarchical and relational data. The system can also be used for extending documents with relational and non-relational data and applying updates based on these documents to relational database targets. The system can also be used for mapping Nested Relational Data to function calls that accept tables as parameters and return multiple scalar and table parameters as output.

#### BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 shows a table that is related to a single row of another table.

Fig. 2 shows the data of Fig. 1, organized as multiple rows in a single table.

Fig. 3 shows the data of Fig. 1, organized as multiple tables related by a join.

Fig. 4 illustrates multiple levels of nested tables contained in one column.

Fig. 5 illustrates a more general example of multiple levels of nested tables contained in more than one column.

Fig. 6 is a block diagram of a database system according to one embodiment of the present invention.

Fig. 7 illustrates schema relating to nested tables; Fig. 7A shows input tables and Fig. 7B shows an output schema.

Fig. 8 illustrates a process of grouping values across nested tables.

Fig. 9 illustrates a process of unnesting data; Fig. 9A shows how a table with a nested table would be unnested into a cross-product of the parent table and a child (nested) table; Fig. 9B illustrates unnesting into separate tables; Fig. 9C illustrates unnesting at multiple levels.

Fig. 10 illustrates a case where unnesting might produce unintended effects.

Fig. 11 graphically illustrates an unnesting process and its effects on a query.

Fig. 12 illustrates a process of converting a DTD to tables.

Fig. 13 illustrates the XML encoding of a DTD definition.

Fig. 14 illustrates various real-time data flows.

Fig. 15 illustrates an operation of joining two inputs in a query.

Fig. 16 illustrates real-time data flows that use supplementary information.

Fig. 17 illustrates data flows depending on cached values.

Fig. 18 illustrates branching data flows based on rules.

Fig. 19 is an illustration of a complex real-time data flow.

Fig. 20 is an illustration of a GUI for specifying a data flow.

Fig. 21 is a block diagram of a schema conversion system.

Figs. 22-26 are tables illustrating various aspects of an NRDM system.

### DESCRIPTION OF THE SPECIFIC EMBODIMENTS

In a specific embodiment a Nested Relational Data Model (NRDM) is designed to support hierarchical and relational components used to represent business data. Business documents are typically hierarchical with multiple repeating sets. For example, an order contains a set of repeating line items. It may also have a set of customers associated with it.

Business documents used to exchange data between software systems within an enterprise or between enterprises need to be represented as complex hierarchical documents. The industry and the research community use well-known representations such as EDI and XML to capture and represent such documents. The system described herein provides methods for mapping such documents to a Nested Relational format, methods for transforming and manipulating of these documents represented using the Nested Relational Data Model, converting such documents to relational format and mapping them to multiple relational tables, and a method of converting the data in a nested relational format back to an external hierarchical format such as XML.

The system provides a method to apply declarative rules to map the hierarchical (e.g., XML or EDI) data to relational tables and vice versa; declarative rules to enrich hierarchical data with data from other relational or hierarchical sources; declarative rules to perform multi-stage transformations. The system allows declarative transformations to be applied to hierarchical data, and the ability to transparently apply rules to heterogeneous databases and files; as well as in the ability to apply multi-stage transformations. Declarative specifications (such as SQL) describe what to do with data, as opposed to procedural specifications (such as C++ code) that described how to do it.

"Nested data" is data in a table that is related to a single row of another table. Sales orders are often presented using nesting: the line items in a sales order are related to a single header. For a table of sales order headers, each row includes its own table of line items. An example of this is shown in Fig. 1. Of course, the same data could  
5 be represented without nested tables. For example, the data could be represented as multiple rows in a single table as shown in Fig. 2, or as multiple tables related by a join as shown in Fig. 3.

One source of data for a nested table is the result of a query using the values in the related row in the parent table. As used herein, "parent table" refers to a  
10 table within which another table is nested and "child table" or "nested table" refers to a table that is nested in a column of a parent table. A nested table is said to have a relationship with the table within which it is nested and where levels are associated with tables, a parent table would have a level that is designated with a number one higher than the child tables nested in that parent table. For example, Fig. 4 shows a parent table 10, a  
15 nested (child) table 12 one level below table 10 and nested tables 14(a)-(b) that are nested in table 12 and are two levels below table 10.

Preferably, a unique instance of each nested table exists for each row at each level of a relationship. As illustrated in Fig. 5, each row at each level can have any  
number of columns containing nested tables.

Fig. 6 shows various aspects of a database system 100 that handles NRDM  
20 data. System 100 is shown comprising a metadata mapper 104 that maps DTD 102 w/hierarchical structures to NRDM schema that are stored in schema storage 106. These components are shown as being part of a preprocessing section, with other portions being part of a real-time section, but it should be understood that all of the process or none of  
25 the processing might be done in real-time without departing from the essence of the invention. Notwithstanding that caveat, the descriptions below reference an example wherein DTDs are converted to NRDM schema and stored and documents are converted by system 100 in real-time after such conversion.

One such real-time process involved a document 110 being passed to an  
30 importer, then to a transformation engine (TE) 114 and an exporter 116 to result in a document in a new format 118 (in some cases, the formats of document 110 and document 118 might be the same, but some transformation has occurred). Document 110 is a structured document, such as an XML document, an HTML page, a document having other structure, or other structured data object.

Importer 112 converts the document into NRDM data so that TE 114 can operate on data in the NRDM space, thus simplifying many transform operations, as described below. TE 114 accepts data in NRDM format as its input and outputs data in NRDM format. Of course, data in NRDM (Nested Relational Data Model) format need not have nested data (for example, if the input data can be structured such that nesting is not needed). Because TE 114 operates on NRDM structures, the transformations performed by TE 114 can be expressed simply as a declarative specification, thus greatly simplifying the process of transforming complex data. In effect, importer 112 converts a hierarchical document into a relational database form to which declarative statements can be applied.

Exporter 116 exports the data in a suitable form, such as XML documents, relational tables or flat files.

#### Data Flows

In a graphical interface used to build data flows and/or nested data structures, such as the ActaWorks™ system developed by Acta, Inc. structures of nested data in input and output schemas of sources, targets, and transforms in data flows are presented to a designer. An example of an input schema 60 is shown in Fig. 7A and an example of an output schema 62 is shown in Fig. 7B. Input schema 60 shows a table A that has columns column1, column2 and a column for a nested table B, which in turn has columns column4 and column5. Input schema 60 also shows a table Z that has columns column11, column12 and a column for a nested table Y, which in turn has columns column14 and column15. In Fig. 7A, and others, nested tables appear with a table icon paired with a plus sign, which indicates that the object contains columns (a minus sign indicates that the object is open and if it has columns, those columns are visible).

In a relational database system (RDS) using a declarative language such as SQL, a query transform might take the form of a SELECT statement that is executed by the RDS. When working with nested data in an nested relational data model (NRDM) system according to some aspects of the present invention, the query can specify SELECTs at each level of a relationship defined in the output schema. Thus, while a SELECT statement might be constrained to include only references to relational data sets, a query that includes nested data might include a SELECT statement to define operations on each table in the output--each context for the input data set is transformed.

In such an NRDM system, the FROM clause descriptions and the behavior of the query are the same with nested data as with relational data, but the new interface of

contexts allows the data flow designer to distinguish multiple SELECTs from each other within a single query. At any context, the FROM clause can contain any top-level table from the input or any table that is a column of a table in the FROM clause of the next higher context.

5           When rows of one table (a child table) are nested inside another table (a parent table), the data set produced in the nested table is the result of a query against the first table using the related values from the second table. For example, if sales information is available as a header table and a line-item table, the sales information can be organized as a parent table of header information and a child table containing line-item data here the line-items are nested under the header table. The line items for a single row of the header table are equal to the results of a query including the order number, as might be found using the following statement:

15       SELECT \* FROM LineItems  
          WHERE Header.OrderNo = LineItems.OrderNo

Correlation can be used to construct a nested table from columns from a higher-level context. In a nested-relational model, the columns in a nested table are implicitly related to the columns in the parent row. To take advantage of this relationship, the parent table can be used in the construction of the nested table. The higher-level column is a correlated column. Including a correlated column in a nested table may serve at least two purposes: 1) the correlated column is a key in the parent table and 2) making the correlated column an attribute in the parent table. Including the key in the nested table allows for the maintenance of you a relationship between the two tables after converting them from the nested data model to a relational model. Including the attribute in the nested table allows for the use of the attribute to simplify correlated queries against the nested data.

Correlated columns can include columns from the parent table and any other tables in the FROM clause of the parent table. If the correlated column comes from a table other than the immediate parent, the data in the nested table includes only the rows that match both the related values in the current row of the parent table and the value of the correlated column.

Values can be grouped across nested tables. Thus, when a statement includes a Group By clause for a table with a nested table, the grouping operation combines the nested tables for each group. For example, to assemble all the line items

included in all the orders for each state from a set of orders, the designer would set the Group By clause in the top-level of the data set to the state column (Order.State) and create an output table that includes State column (set to Order.State) and LineItems nested table. The result of such an operation might result with the table shown in Fig. 8. The result is a set of rows (one for each state) that has the State column and the LineItems nested table that contains all the LineItems for all the orders for that state.

Nested data can also be unnested. When loading a data set that contains nested tables into a relational (non-nested) target, the nested rows will be unnested. Take, for example, a message containing a sales order that uses a nested table to define the relationship between the order header and the order line items. To load the data into relational tables, the multi-level must be unnested. Unnesting a table produces a cross-product of the top-level table (parent) and the nested table (child), as shown in Fig. 9A. Different columns from different nesting levels might be loaded into different tables. A sales order, for example, may be flattened so that the order number is maintained separately with each line item and the header and line item information loaded into separate tables, as shown in Fig. 9B.

Any number of nested tables can be unnested at any depth. No matter how many levels are involved, the result of unnesting tables is a cross product of the parent and child tables. When more than one level of unnesting occurs, the inner-most child is unnested first, then the result--the cross product of the parent and the inner-most child--is then unnested from its parent, and so on to the top-level table, creating the result shown in Fig. 9C.

Unnesting all tables (cross product of all data) may not produce the results intended. For example, if multiple customer values are included in an order, such as ship-to and bill-to addresses, flattening a sales order by unnesting customer and line item tables produces rows of data that may not be useful for processing the order. This is illustrated in Fig. 10. Using the GUI, the specification of the data flow is shown in Fig. 11.

A DTD (document type definition) describes the data schema of an XML message or file. Real-time data flows read and write XML messages based on a specified DTD format. One DTD can describe multiple XML sources or targets. Batch data flows can read and write data to files based on a specified DTD format.

DTDs can be imported into the NRDM system, either directly or by importing an XML document that contains a DTD. During import, the NRDM system

converts the structure defined in the DTD into an internal nested-relational data model. Elements below the root-level that contain other elements become nested tables and elements that do not contain other elements become columns. Attributes become columns in the corresponding element's schema.

5                   The NRDM system applies the following rules to convert the DTD to tables, columns, and nested tables:

- Any element that contains PCDATA only and no attributes becomes a column.
- Any element with attributes or other elements (or in mixed format) becomes a table.
- An attribute becomes a column in the table corresponding to the element it supports.
- 10 - Any occurrence of choice operators is converted to strict ordering.
- Any occurrence of optional operators is converted to strict ordering.
- Any occurrence of ()\* or ()+ becomes a table with an internally generated name--an implicit table.

15                   After these rules have been applied, the NRDM system optimizes the format using two more rules, except where doing so would allow more than one row at the root element:

- If an implicit table contains one and only one nested table, then the implicit table can be eliminated and the nested table can be attached directly to the parent of the implicit table. For example, the SalesOrder element might be defined as follows in the DTD:

20                   <!ELEMENT SalesOrder (Header, LineItems\*) >

25                   When converted, the LineItems element with the zero or more operator would become an implicit table under the SalesOrder table. The LineItems element itself would be a nested table under the implicit table, as shown in Fig. 12A. Because the implicit table contains one and only one nested table, the format would be optimized to remove the implicit table, as shown in Fig. 12B.

- If a nested table contains one and only one implicit table, then the implicit table can be eliminated and its columns placed directly under the nested table. For example, the nested table LineItems might be defined as follows in the DTD:

30                   <!ELEMENT LineItems (ItemNum, Quantity)\*>

35                   When converted, the grouping with the zero or more operator would become an implicit table under the LineItems table. The ItemNum and Quantity elements would become columns under the implicit table, as shown in Fig. 12C. Because the

LineItems nested table contained one and only one implicit table, it would be optimized to remove the implicit table, as shown in Fig. 12D.

If the DTD contains an element that uses an ancestor element in its definition, the definition of the ancestor can be expanded for a fixed number of levels.

5 For example, given the following definition of element "A":

A: B, C  
B: E, F  
F: A, H

10 The system produces a table for the element "F" that includes an expansion of "A." In this second expansion of "A," "F" appears again, and so on until the fixed number of levels. In the final expansion of "A," the element "F" appears with only the element "H" in its definition.

#### Real-Time Sources

15 A real-time source in a real-time data flow determines the message that the real-time data flow will process. The source object represents the schema of the expected messages. Messages received are fit to the schema. Real-time data flows accept real-time source types such as Extensible Markup Language formatted (XML) messages or intermediate documents, such as IDocs published from an SAP R/3 application server.

20 The format of the XML message is specified by a document type definition (DTD). The DTD describes the schema of data contained in the message and the relationships among the elements in the data. For a message that contains information to place a sales order--order header, customer, and line items--the corresponding DTD includes the order structure and the relationship between data, as shown by the example in Fig. 13.

25 The following examples provide a high-level description of how real-time data flows address typical real-time scenarios. Fig. 14A shows a real-time data flow as might be used to load transactions into an ERP system, such as SAP R/3. A real-time data flow can receive a transaction from an electronic commerce application and load it to an ERP system. Using a query transform, one can include values from a data warehouse to supplement the transaction before applying it against the ERP system.

30 Fig. 14B shows a real-time data flow for collecting ERP data into a warehouse. Real-time data flows can receive messages from the ERP through IDocs. Each IDoc contains a transaction that the real-time data flow can load into a data

warehouse or a data mart. In this way, IDocs can be used to keep the data in a warehouse current.

Fig. 14C shows a real-time data flow for retrieving values from a cache or and ERP system. This allows for real-time data flows that use values from a data warehouse to determine whether or not to query the ERP system directly.

#### Supplementary Sources

When more data is needed than what is provided in the content of a message to complete the message processing, supplementary sources might be used. For example, processing a message that contains a sales order from an electronic commerce application that contains the customer name might require that, when the order is applied against your ERP system, more detailed customer information is needed. Inside the real-time data flow, the message is supplemented with the customer information to produce the complete document to send to the ERP system. The supplementary information may come from the ERP system itself or from a cache containing the same information cached. Examples of such data flows are shown in Figs. 15, 16A, 16B.

Tables and files (including XML files) as sources in real-time data flows can provide this supplementary information. The real-time data flow extracts data from the supplementary source as indicated by the logic defined in the real-time data flow.

Tables or files that are used as sources and have a cache option allow for the data extracted to be stored in memory until the data flow processing is complete. In real-time data flows, sources should not be cached unless the data being cached is small and is unlikely to be updated in the life of the real-time data flow.

In batch data flows, caching can improve the performance of data flow processing by reducing the number of times a set of data is read from the database or file source. In real-time data flows, however, the improvement in performance provided by caching is minimized by the likelihood that the real-time data flow reads only a small amount of data from the source for any given message. In addition, because the real-time data flow reloads cached data only when an access server shuts it down and restarts it, cached data may become stale in memory.

Tables can be sources in real-time data flows after their metadata is imported into the repository. When the real-time data flow starts, it opens a connection to the source database. This connection remains open as long as the real-time data flow is running. If a table is included in a join with a real-time source, the data set from the real-time source is included as the outer loop of the join.

R/3 tables can be sources in real-time data flows after their metadata is imported into the repository. When the real-time data flow performs a query against the R/3 table, it executes an R/3 function call to extract the data through the SAP R/3 application server. This method of extracting data from SAP R/3 is particularly well suited to extracting a small amount of specific data (on the order of 1 to 10 rows) in a real-time system, but might not work well as a substitute to using R/3 data flows to produce ABAP programs to extract large amounts of data in a batch system.

Data from XML files can be used as sources in real-time data flows, if a DTD that describes the data in the file is imported.

#### 10 Supplementing Message Data

The data included in messages from real-time sources may not map exactly to requirements for processing or storing the information. If not, steps can be defined in the real-time data flow to supplement the message information. One technique for supplementing the data in a real-time source includes these steps in a real-time data flow:

1. Include a table or file as a source. In addition to the real-time source, include the files or tables that supply the supplementary information.
2. Use a query to extract needed data from the table or file. Use the data in the real-time source to find the needed supplementary data. A join expression can be used in the query so that only the specific values required from the supplementary source are extracted.

Fig. 16A shows an example where a message includes sales order information with the ultimate goal to return order status. In this case, the business logic uses the customer number and priority rating to determine the level of status to return. The message includes only the customer name and the order number. The real-time data flow is then defined to retrieve the customer number and rating from other sources before determining the order status.

A real-time data flow might include logic to determine when responses can be generated from data in a cache and when they must be generated from data in an ERP system. One technique for constructing this logic includes the steps in the real-time data flow (illustrated in Figs. 17-20):

1. Determine the rule for when to access the cache and when to access the ERP system.
2. Compare data from the real-time source with the rule.
3. Define each path that could result from the outcome. Consider the case where the rule indicates ERP access, but the ERP system is not currently available.
4. Merge the results from each path into a single data set.
5. Route the single result to the real-time target.

This example describes a section of a real-time data flow that processes a new sales order. The section is responsible for checking the inventory available of the ordered products--it finds an answer to the question, "is there enough inventory on hand to fill this order?" The rule controlling access to the ERP system indicates that the inventory (Inv) must be more than a pre-determined value (IMargin) greater than the ordered quantity (Qty) to consider the cached inventory value acceptable. The comparison is made for each line item in the order.

Fig. 18 illustrates a branch in the data flow based on a rule. An XML source contains the entire sales order, yet the data flow compares values for line items inside the sales order. The XML target that ultimately returns a response requires a single row at the top-most level. Because this data flow needs to be able to determine inventory values for multiple line items, the structure of the output requires the inventory information to be nested. The input is already nested under the sales order; the output can use the same convention. In addition, the output needs to include some way to indicate that the inventory is or is not available.

Fig. 19 illustrates several ways to return values from the ERP. For example, a lookup function or a join on the specific table could be used in the ERP system. The example uses a join so that the processing can be performed by the ERP system rather than the NRDM system. As in the previous join, if a value might not be returned by the join, an outer join can be defined so that the line item row is not lost.

Fig. 20 illustrates a GUI used to specify transformations and a specific transformation specified with that GUI.

Fig. 21 is a block diagram of a schema converter. In the example shown, an NRDM schema is converted to a DTD schema.

### Other Uses

One of the advantages of operating a transformation engine on NRDM data structures, as described above, is that the transformation engine can operate on hierarchical data as if it were a relational table. Thus, hierarchical documents, such as XML documents can be operated on using declarative statements, such as SQL, regardless of how many levels of hierarchy are present. One method of effecting such a benefit is to nest child tables into columns of parent tables and use a transformation engine that handles NRDM data as its input and as its output. The transformation engine can be sandwiched between an importer that converts hierarchical documents into NRDM data structures and an exporter that generates hierarchical documents from NRDM data structures.

There are various ways to implement NRDM data structures. For example, conventional relational tables can be used, where a column of the parent table stores a pointer to a child table. A separate child table could exist for each row of the parent table that does not have a NULL value for that row and column, or where the child tables for each row have corresponding formats, the data representing the child tables could be implemented as subtables of one child data-holding table. Regardless of the underlying structure, the transformation engine deals with the data structures as nested tables and applies declarative statements accordingly.

Other aspects of the system described herein might find uses apart from NRDM data structures and systems. For example, requests received from applications for data processing and/or transformation might operate on nested tables, but might also operate on conventional relational tables.

The applications often provide application programming interfaces (APIs) through which other programs interact with the application. Often, the designer of a program that interacts with the application must know the interfaces and correctly specify the parameters of a particular function call. However, some applications might accept as an input NRDM data or a hierarchical document. In some cases, the application interface could be such that the semantics of the function call are in a document submitted as a parameter and then one generic interface is all that is needed to call the application.

### Example Implementation

An example of an NRDM system according to various aspects of the present invention will now be described. It should be understood that the invention is not limited to this specific example. The example system supports hierarchical data models

such as IDoc and XML and provides for a hierarchical structure to support a hierarchical data model represented as a single row that contains scalar columns and repeating group(s) of embedded rows forming nested table(s), where nesting can be arbitrarily deep and an implicit relationship is not required between embedded rows and parent (i.e., the children rows do not need to contain a key to join it back to the parent row).

The NRDM system can capture an entire business transaction in a single hierarchical structure and transform a hierarchical structure as a single entity using relation operators that can be applied at any level of the hierarchy. A hierarchical structure when applied as a single database transaction can be loaded to a set of tables belonging to a single datastore.

#### Data Model

In NRDM, the first normal form requirement that a column be a scalar is removed. In NRDM, a column can be a scalar or a relation value, which we refer to as a nested table. A scalar column definition has a name, type (including length, precision, domain info, etc.) and, at run time, contains either a value or a NULL indicator. A nested table definition has a name, schema (e.g., a list of column definitions) and, at run time, contains either one or more rows of the schema specified in the nested table definition or an empty table indicator (e.g., ISEMPY).

#### DDL Operations

AL\_NESTED\_TABLE is used below to define a nested table for DDL operations. For example, creating a view with nested table might be done by the following statements:

```
CREATE VIEW V1 (
  ORDER_ID INT,
  PROD_INFO AL_NESTED_TABLE (
    PROD_ID INT,
    QTY INT,
    VENDOR_INFO AL_NESTED_TABLE (VNDR_ID CHAR(5),
                                VNDR_CITY CHAR(65))
  ),
  CID INT,
  CCITY CHAR(65)
);
```

Fig. 22 illustrates a data table that might result for the above statements.

#### DML Operations

Relational operations such as select, project, etc. can be used on NRDM data. Nested relations can be accessed as regular relations in the context (scope) of their parents. In other words, wherever a scalar column is used, a nested table can be used. If a parent table is used in a FROM clause, all the nested tables can be used in the SELECT

and WHERE clauses and nested subqueries as full-fledged tables. If two parent tables having a same name for a nested table are used in a relational operation, the nested tables should be qualified with the parent tables.

Nested subqueries allow for accessing and transforming data inside nested relations. Nested subqueries can transform data in nested relations, nest, unnest and join data in nested relations with the data in its parents and handle operations such as ISEMPY, AL\_NEST, AL\_NEST\_SET and AL\_UNNEST for NRDM data. The AL\_NEST operator creates partitions based on the formation of equivalence classes to generate nested tables. It operates on a row basis. AL\_NEST\_SET operator is similar to AL\_NEST but operates on a set basis. The AL\_UNNEST operator transforms a relation into one, which is less deeply nested by concatenating each tuple in the relation being unnested to the remaining attributes in the relation.

The AL\_NEST operator creates partitions based on the formation of equivalence classes to generate nested tables. Two tuples are equivalent if they have the same values for attributes, which are not being nested. AL\_NEST operates on a row basis. Nesting can be done in two ways using a user interface (such as the GUI described above). A nested table can be dragged from the input to the output of a query transform and placed at the same or deeper level, or a nested schema can be created and columns from the input can be dragged and dropped into the newly created schema.

An explicit FROM clause might be needed where two views are coming into a query transform, and columns are selected from only one the views. The generated language is to select from both the views. For nesting of two input views containing only scalar columns, selecting from the both the views at the same level might not be desired. The following example illustrates this. Given a flat view V1 as:

```
25      CREATE VIEW ORDERS (ORDER_ID INT, PROD_ID INT, QTY INT,
                          CID INT, CCITY VARCHAR(65))
```

```
      CREATE VIEW VENDORS (PROD_ID INT, VNDR_ID VARCHAR(5),
                          VNDR_CITY VARCHAR(65))
```

the table of flat relations shown in Fig. 23 results. A two level nesting to include vendor information using a JOIN can be demonstrated by the following example:

```
35      CREATE VIEW V2 (ORDER_ID INT,
                      PROD_INFO AL_NESTED_TABLE (PROD_ID INT,
                                                  QTY INT,
                                                  VENDOR_INFO
                                                  AL_NESTED_TABLE (
                                                  VNDR_ID CHAR(5),
                                                  VNDR_CITY CHAR(65))
```

```

    ),
    CID,
    CCITY
  )
5  AS SELECT ORDER_ID,
    AL_NEST (CREATE VIEW PROD_INFO (PROD_ID INT, QTY INT)
    AS SELECT PROD_ID,
    QTY,
10  AL_NEST (CREATE VIEW VENDOR_INFO
    (VNR_ID CHAR(5),
    VNR_CITY CHAR(65)) AS
    SELECT VNR_ID, VNR_CITY
    FROM VENDORS
    WHERE VENDORS.PROD_ID = L1.PROD_ID
15  )
    AS VENDOR_INFO
    FROM ORDERS L1
    WHERE L1.ORDER_ID = L0.ORDER_ID AND
20  L1.CID = L0.CID AND
    L1.CCITY = L0.CCITY
  )
  AS PROD_INFO,
  CID,
25  CCITY
  FROM ORDERS L0

```

The explicit FROM clause prevents the usage of the VENDORS in the outermost select. This may produce a nested table as shown in Fig. 22, except with three rows with ORDER\_ID equal to 100, two rows with ORDER\_ID equal to 200 and one row with ORDER\_ID = 300, because AL\_NEST operates on a row basis, which can produce duplicates.

The AL\_NEST operator may be used to perform nesting on a set of rows also. If there is a GROUP BY, the set formed by the GROUP BY is used. If there are aggregate functions and a GROUP BY is specified, the set formed by the GROUP BY is used. If there are aggregate functions and a GROUP BY is not specified, then the default grouping is the entire table. All nested tables in the set operated by the AL\_NEST may be merged.

#### Using AL\_NEST SET with an Aggregate Function

This operation may take in a view with nested tables and produce a single row, which has count of ORDER\_ID's and the merge of all nested tables:

```

40  CREATE VIEW V2 (NUM_ORDERS INT,
    PROD_INFO AL_NESTED_TABLE (PROD_ID INT,
    QTY INT
45  )
  )
  AS SELECT COUNT(ORDER_ID),
    AL_NEST_SET (CREATE VIEW PROD_INFO (PROD_ID INT,
    QTY INT) AS

```

```

SELECT PROD_ID, QTY
FROM PROD_INFO
)
AS PROD_INFO,

```

5

FROM V1

Such a query might produce the table shown in Fig. 24. If the nested table(s) SELECT(S) have WHERE clauses, the nested table(s) might first be merged and the filters applied to the merged table(s).

## 10 AL\_UNNEST

The AL\_UNNEST operator transforms a relation into one that is less deeply nested by concatenating each tuple in the relation being unnested to the remaining attributes in the relation. To unnest the vendor information from the nested table in Fig. 22, the following ATL might be defined:

```

15 CREATE VIEW V2 (ORDER_ID INT,
    PROD_INFO AL_NESTED_TABLE (PROD_ID INT,
                                QTY INT,
                                VNDR_ID CHAR(5)))
    AS SELECT ORDER_ID,
20    AL_NEST (CREATE VIEW PROD_INFO (PROD_ID INT, QTY INT) AS
        SELECT V1.PROD_INFO.PROD_ID,
              V1.PROD_INFO.QTY,
              AL_UNNEST (CREATE VIEW VDR_INFO
25                          (VNDR_ID INT) AS
                          SELECT
                          V1.PROD_INFO.VENDOR_INFO.VNDR_ID
                          FROM V1.PROD_INFO.VENDOR_INFO)
        FROM V1.PROD_INFO)
    AS PROD_INFO
30 FROM V1

```

WHERE clauses can be applied in the SELECT for unnesting by drilling into the nested table which would produce a query transform, specifying the condition there, as shown in the following example:

```

35 CREATE VIEW V2 (VNDR_ID CHAR(5), VNDR_CITY CHAR(65))
    AS SELECT DISTINCT AL_UNNEST (CREATE VIEW
        UNEST1 (VNDR_ID CHAR(5),
              VNDR_CITY CHAR(65))
        AS SELECT
40    AL_UNNEST (CREATE VIEW
        UNEST2 (VNDR_ID CHAR(5),
              VNDR_CITY
              CHAR(65))
        AS SELECT VNDR_ID, VNDR_CITY
        FROM VENDOR_INFO)
        FROM PROD_INFO
45    )
    FROM V1

```

## Project

An example of a simple projection from one hierarchical structure to another would be:

50

```

CREATE VIEW V2 (
    ORDER_ID INT,
    PROD_INFO AL_NESTED_TABLE (PROD_ID INT, QTY INT)
)
5  AS SELECT ORDER_ID,
    AL_NEST (CREATE VIEW PROD_INFO (PROD_ID INT, QTY INT)
    AS SELECT V1.PROD_INFO.PROD_ID, V1.PROD_INFO.QTY
        FROM V1.PROD_INFO)
    AS PROD_INFO
10 FROM V1

```

The qualifier V1.PROD\_INFO in the nested relation is not really needed; the nested query could have been written using just PROD\_INFO. The result might be the table shown in Fig. 25.

#### Select

15 Filter conditions can be applied at various levels. Consider the example of view V1 (Fig. 22) that has three levels of nesting. A filter on the nested relation PROD\_INFO might be implemented as follows:

```

CREATE VIEW V3 (ORDER_ID INT,
    PROD_INFO AL_NESTED_TABLE (PROD_ID INT, QTY INT)
)
20 AS SELECT
    ORDER_ID,
    AL_NEST (CREATE VIEW PROD_INFO (PROD_ID INT, QTY INT)
    AS SELECT V1.PROD_INFO.PROD_ID,
        V1.PROD_INFO.QTY
        FROM V1.PROD_INFO
        WHERE V1.PROD_INFO.QTY > 50)
    AS PROD_INFO
30 FROM V1

```

This may select all the rows from V1, but for the nested table PROD\_INFO, only those rows are chosen where the quantity ordered QTY is greater than 50, resulting in the table shown in Fig. 26.

#### Alternate Support For Filters In The WHERE Clause

35 For a nested table to be used in a WHERE clause sub-query, support within a WHERE clause should be available. If such support is not available, it can be overcome by using two stages and the ISEMPY operator for nested tables. Nested tables can be used in a WHERE clause only with the ISEMPY operator. The following example illustrates the use, selecting all the rows from V1 that have ORDER\_ID greater

```

40 than 100 and that have at least one product with a quantity ordered greater than 50.
    CREATE VIEW V3 (ORDER_ID INT,
        PROD_INFO AL_NESTED_TABLE (PROD_ID INT, QTY INT),
        TEMP_PROD_INFO AL_NESTED_TABLE (PROD_ID INT, QTY
45 INT)
    )
    AS SELECT
        ORDER_ID,

```

```

5      AL_NEST(CREATE VIEW PROD_INFO(PROD_ID INT, QTY INT)
        AS SELECT V1.PROD_INFO.PROD_ID,
          V1.PROD_INFO.QTY
        FROM V1.PROD_INFO
        )
      AS PROD_INFO,
      AL_NEST(CREATE VIEW PROD_INFO(PROD_ID INT, QTY INT)
10     AS SELECT V1.PROD_INFO.PROD_ID,
          V1.PROD_INFO.QTY
        FROM V1.PROD_INFO
          WHERE V1.PROD_INFO.QTY > 50)
      AS TEMP_PROD_INFO

15     FROM V1 WHERE V1.ORDER_ID > 100

      CREATE VIEW V4 (ORDER_ID INT,
        PROD_INFO AL_NESTED_TABLE(PROD_ID INT, QTY INT)
        )
      AS SELECT
20     ORDER_ID,
      AL_NEST(CREATE VIEW PROD_INFO(PROD_ID INT, QTY INT)
        AS SELECT V1.PROD_INFO.PROD_ID,
          V1.PROD_INFO.QTY
        FROM V1.PROD_INFO
        )
25     AS PROD_INFO

      FROM V3 WHERE !ISEMPTY(TEMP_PROD_INFO)

```

Join

30 Nested relations can be joined with any other relations. An example is  
given below:

```

      CREATE VIEW ORDERS (ORDERID INT, PRODUCTS
        AL_NESTED_TABLE (PRODID INT, PRODNAME VARCHAR (10)));

35     CREATE VIEW VENDORS (PRODID INT, VENDORID INT,
          VENDORNAME VARCHAR (10));

      CREATE VIEW ORDERS_WITH_VENDORS (ORDERID INT,
        PRODUCTS AL_NESTED_TABLE (PRODID INT,
40          PRODNAME VARCHAR (10),
          VENDORID INT)

      AS
      SELECT ORDERID,
        AL_NEST (CREATE VIEW PRODUCTS (PRODID INT,
45          PRODNAME VARCHAR (10),
          VENDORID INT)

          AS SELECT PRODID, PRODNAME, VENDORID
            FROM PRODUCTS, VENDORS
              WHERE PRODUCTS.PRODID = VENDORS.PRODID)

50     AS PRODUCTS
      FROM ORDERS GROUP BY ORDERID

```

### Nested Table Transform

A system transform is available that takes in a flat view and produces a singleton that has a N integer scalar column with a value 1, and a nested table containing the input view.

#### 5 Tables as Parameters

Tables can be used as parameters for imported functions. Given a function `get_orders` with an input parameter `customer_id` and an output parameter `orders`:

```
10 CREATE FUNCTION get_orders (cust_id int,
                                orders AL_NESTED_TABLE(order_id int, ...)
                                OUTPUT,
                                cust_info AL_NESTED_TABLE(cust_name, ...)
                                OUTPUT);
```

Get orders for each customer by calling the orders function:

```
15 CREATE VIEW customer_orders (customer_id int,
                                orders AL_NESTED_TABLE (order_id
                                                            int, ...))
AS SELECT customer_id,
           AL_NEST (get_orders (customer_id)::orders)
20           AS orders
FROM customers;
```

if the function has multiple tables as outputs, and all or some of them are required, then the function has to be invoked multiple times: once for each output.

```
25 CREATE VIEW customer_orders(customer_id int,
                                cust_info AL_NESTED_TABLE(cust_name, ...),
                                orders AL_NESTED_TABLE (order_id
                                                            int, ...))
AS SELECT customer_id,
           AL_NEST (get_orders (customer_id)::cust_info) AS
30           cust_info
           AL_NEST (get_orders (customer_id)::orders) AS orders
FROM customers;
```

As an optimization, the system could invoke the function only once and use those results for different instances within the query transform. For mapping a function returning table, a user would create a nested table column and map the nested table column to the function returning a table. The schema of the nested table may then be identical to the schema returned by the function. This is a concept of a "generated table". The schema definition of generated table cannot be modified, and it should disappear when the function is removed from the mapping. It should be represented differently in the UI so that a user can distinguish between a generated table and a non-generated table.

### Hierarchical File Reader

A hierarchical file reader reads data generated by data flows that have functions that return tables. There are two main alternatives: model the file reader as an

XML file reader or model the file reader using a proprietary format to represent hierarchical data.

#### Effect of NRDM on System Transforms

5      System transforms such as Table\_Comparison, Hierarchy\_Flattening, etc. accept only rows with scalar columns.

Table\_Comparison: The output schema of the table comparison transform is a generated schema and is same as the schema of the table being compared against. This transform may silently ignore columns that are nested tables.

10      History Preserving: The output schema of the history preserving transform is same as the input schema, and this transform may preserve history only scalar columns and may act as pass through for columns that are nested tables.

Effective Date: The transform may act as pass through for columns that are nested tables.

15      Key Generation: The output schema of the key generation transform is same as the input schema, and this transform may act as pass through for columns that are nested tables.

Map Operation: The output schema of the map operation transform is same as the input schema, and this transform may not allow operations to be mapped for columns as nested tables and may act as pass through for them.

20      Hierarchy Flattening: Columns as nested tables cannot be a parent or child column of a hierarchy, but they can be dragged and dropped attribute columns and thus can appear in the output schema.

Pivot: The output schema of the hierarchy flattening transform is a generated schema and columns, as nested tables may be ignored.

#### 25      A Case Study

    A case study of a Sales Order IDoc using NRDM was performed. The IDoc was captured in a NRDM and perform transformations, to arrive at the same result as if the NRDM was not used, but with simplified specification of the transformations.

30      An IDoc is divided into a control record, data records and a status record. Each control record and status record has numerous fields. For our purpose of validating the NRDM, we treated control records and status records as single varchar columns. The ATL to represent a Sales Order (some of the columns associated with nested tables might be omitted in the listing) is:

```

CREATE VIEW V1 (
  CONTROL_RECORD VARCHAR (100),
  STATUS_RECORD VARCHAR (100),
  E2CMCCO AL_NESTED_TABLE (
5      ZEITP VARCHAR (2), ...,
      E2CVBUK AL_NESTED_TABLE (
          SUPKZ VARCHAR (1), ...,
          E2CVBAK AL_NESTED_TABLE (
10              SUPKZ VARCHAR (1), ...,
              E2CVBK0 AL_NESTED_TABLE (
                  SUPKZ VARCHAR (1),
              ),
              E2CVBP0 AL_NESTED_TABLE (
                  SUPKZ VARCHAR
15                      (1),
              ),
              E2CVBAP AL_NESTED_TABLE (
                  SUPKZ VARCHAR (1),
                  E2CVBA2
20      AL_NESTED_TABLE(
          SUPKZ
          VARCHAR(1),
          ),
          E2CVBUP
25      AL_NESTED_TABLE(
          SUPKZ
          VARCHAR(1),
          ),
          E2CVBPF
30      AL_NESTED_TABLE(
          SUPKZ
          VARCHAR (1)
          ),
          E2CVBKD
35      AL_NESTED_TABLE(
          SUPKZ
          VARCHAR (1),
          ),
          E2CKONV
40      AL_NESTED_TABLE(
          SUPKZ
          VARCHAR (1),
          ),
          E2CVBPA
45      AL_NESTED_TABLE(
          SUPKZ
          VARCHAR (1),
          ),
          E2CVBFA
50      AL_NESTED_TABLE(
          SUPKZ
          VARCHAR (1),
          ),
          E2CFPLT
55      AL_NESTED_TABLE(
          SUPKZ
          VARCHAR (1),
          ),
          E2CVBEP
60      AL_NESTED_TABLE(

```

```

SUPKZ
VARCHAR (1),
),
), # E2CVBAP
), # E2CVBAK
), # E2CVBUK
) # E2CMCC0
) # V1

```

10 The ATL corresponding to the population of the sales order fact table from the above view may be (with some columns omitted for illustration purposes):

```

15 CREATE VIEW V2 ( SO_NUM, # VBAK.VBELN
SOLD_TO, # VBAK.KUNNR
LINE_ITEM_ID, # VBAP.POSNR
CREATE_DATE, # VBAP.ERDAT
SHIP_TO, # VBPA.KUNNR
DELIVERY_STATUS # VBUP.LFGSA
)
20 AS SELECT AL_UNNEST
(SELECT AL_UNNEST
(SELECT AL_UNNEST
(SELECT VBELN, KUNNR,
AL_UNNEST (SELECT POSNR, ERDAT,
25 AL_UNNEST (SELECT KUNNR FROM
E2CVBPA WHERE PARVW =
'WE'),
AL_UNNEST (SELECT LFGSA FROM
30 E2CVBUP) FROM E2CVBAP
)
FROM V1.E2CMCC0.E2CVBUK.E2CVBAK
)
35 FROM V1.E2CMCC0.E2CVBUK
)
FROM V1.E2CMCC0
)
FROM V1
40

```

WHAT IS CLAIMED IS:

1                   1. An apparatus for processing data representable in a hierarchical form,  
2 the apparatus comprising:  
3       an importer having inputs to receive a schema and a structured document from a data  
4       source, wherein the importer outputs a first nested relational data model  
5       (NRDM) data structure representing the structured document according to the  
6       received schema;  
7       an transformation engine that is capable of transforming the first NRDM data  
8       structure output by the importer into a second NRDM data structure according to  
9       a declarative specification of a transform; and  
10       an exporter having an input to receive the second NRDM data structure, wherein the  
11       exporter outputs a transformed hierarchical document in a data structure other  
12       than an NRDM data structure in a form suitable for a data target.

13                   2. The apparatus of claim 1, further comprising means for converting  
14 relational data to an NRDM data structure by vertically partitioning a relation and nesting  
15 parts of the relational data as a nested table.

16                   3. The apparatus of claim 1, further comprising means for converting  
17 nested relational data to relational data by unnesting the nested tables using a  
18 cross-product between a parent row and a child subtable.

19                   4. The apparatus of claim 1, further comprising means for performing a  
20 grouping operation on a nested table that generates a resulting nested table containing a  
21 union of all the nested tables grouped by the operation.

22                   5. The apparatus of claim 1, further comprising means for performing  
23 multi-step transformations, wherein an input to a transformation is results of a previous  
24 transformation, a data source, or both.

25                   6. The apparatus of claim 1, wherein the transformation engine operates on  
26 rules that are applied to data independent of data format.

27                   7. The apparatus of claim 1, wherein the exported is adapted to output one  
28 or more of an XML file, a relational table or a flat file.

- 29                   8. A metadata mapper comprising:  
30           an input for receiving a document description for hierarchical documents; and  
31           an output for outputting an NRDM data structure representing the document  
32           description.
- 33                   9. An apparatus for transforming data representable in a hierarchical form,  
34   the apparatus comprising:  
35           an importer having inputs to receive a schema and a structured document from a data  
36           source, from a data transformer, or from both, wherein the importer outputs a  
37           first nested relational data model (NRDM) data structure representing the  
38           structured document according to the received schema;  
39           an transformation engine that is capable of transforming the first NRDM data  
40           structure output by the importer into a second NRDM data structure according to  
41           a declarative specification of a transform; and  
42           an exporter having an input to receive the second NRDM data structure, wherein the  
43           exporter outputs a transformed hierarchical document in a data structure other  
44           than an NRDM data structure in a form suitable for a data target.
- 45                   10. A method for providing data to an application through a data platform  
46   in a computer system in response to request from the application, the method comprising:  
47           accepting declarative rules for accessing the data from data sources and declarative  
48           rules for transforming the data into a format requested by the application;  
49           mapping relational and non-relational data sources to an NRDM data structure;  
50           interpreting a request;  
51           retrieving data from the data sources;  
52           transforming the data according to the declarative rules; and  
53           returning the transformed data to the application.
- 54                   11. The method of claim 10, wherein requests are processed as messages  
55   and request messages contain sufficient information to drive data extraction into a  
56   data-oriented interface.
- 57                   12. The method of claim 10, wherein the requests are application  
58   programming interface function calls.

59                   13. A method for updating a plurality of data targets from a message,  
60 comprising:

61       making an update request through a data-oriented interface;  
62       specifying declarative rules for updating the data targets;  
63       importing metadata that maps relational and non-relational data targets to NRDM  
64       data structures;  
65       interpreting incoming update requests;  
66       transforming the data according to the declarative rules; and  
67       updating the data targets.

68                   14. The method of claim 13, further comprising:  
69       making an update request using an application; and  
70       causing one of a response to be sent to the application, an update of data, or both.

71                   15. The method of claim 13, further comprising a step of combining the  
72       update request with other data before updating the data targets.

73                   16. A method of providing input to an application expecting one or more  
74       tables as parameters to an input message, the method comprising:  
75       mapping data in a NRDM data structure to function parameters; and  
76       making a function calls to the application using the NRDM mapped data structure.

Order data set

OrderNo	CustID	ShipTo1	ShipTo2	LineItems
9999	1001	123 State St.	Town, CA	

Item	Qty	ItemPrice
001	2	10
002	4	5

FIG. 1

Order header data set

OrderNo	CustID	ShipTo1	ShipTo2
9999	1001	123 State St.	Town, CA

Order data set

OrderNo	CustID	ShipTo1	ShipTo2	Item	Qty	ItemPrice
9999	1001	123 State St.	Town, CA	001	2	10
9999	1001	123 State St.	Town, CA	002	4	5

Line-item data set

OrderNo	Item	Qty	ItemPrice
9999	001	2	10
9999	002	4	5

WHERE  
Header.OrderNo = LineItem.OrderNo

FIG. 2 (PRIOR ART)

FIG. 3 (PRIOR ART)

Order data set

OrderNo	CustID	ShipTo1	ShipTo2	LineItems
9999	1001	123 State St.	Town, CA	

Item	ItemQty	ItemPrice
001	2	
002	4	

Qty	Sell Price
1	25
20	23

Qty	SellPrice
1	200
10	190

FIG. 4

14 (a)

Order data set

OrderNo	LineItems	CustInfo
9999		

Item	ItemQty	ItemPrice
001	2	
002	4	

Qty	Sell Price
1	25
20	23

Qty	SellPrice
1	200
10	190

Type	CustID	Contacts
Ship	1001	
Bill	7777	

Name	Phone
Alvarez	555-1234
Tanaka	555-6666

Name	Phone
Samp	333-1234

FIG. 5

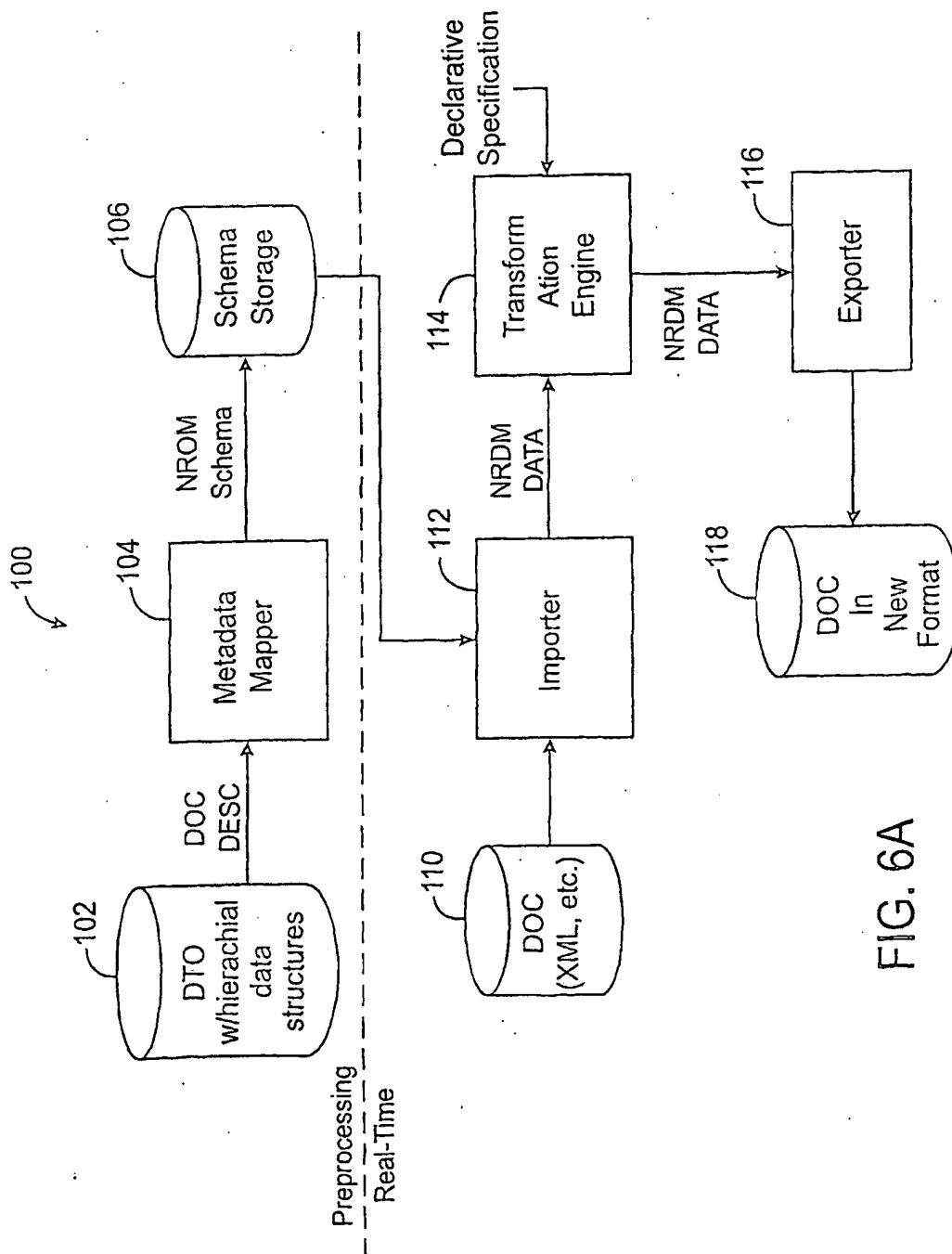


FIG. 6A

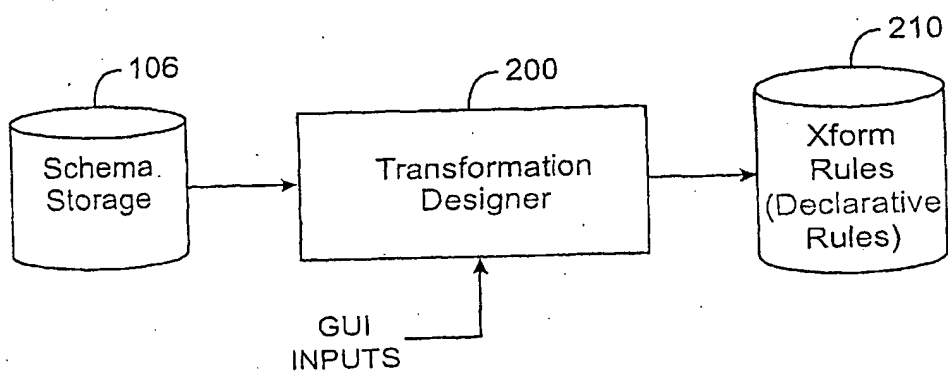


FIG. 6B

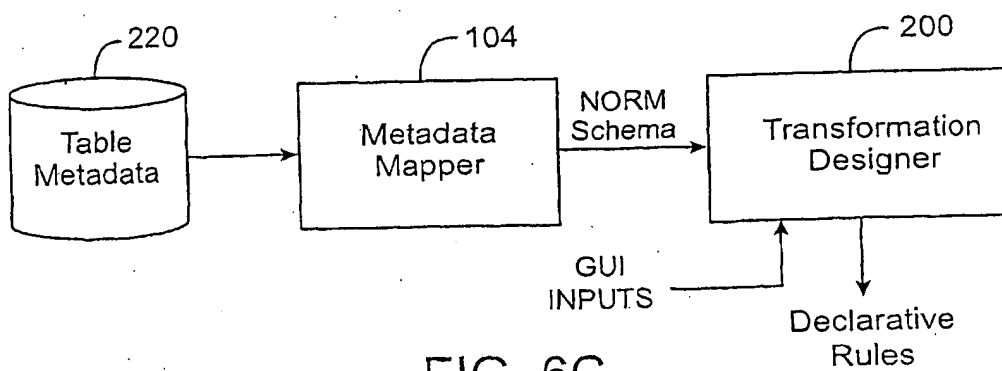
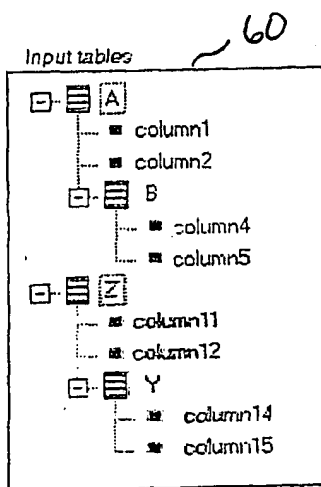


FIG. 6C

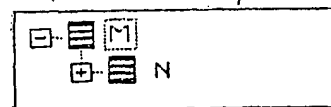


Correct combination of FROM clauses  
for the two contexts in the output:

```

SELECT columns FROM A
SELECT columns FROM A,B, Z
  
```

Output schema 62



Incorrect combination of FROM clauses  
for the two contexts in the output:

```

SELECT columns FROM A
SELECT columns FROM A,B, Z,Y
  
```

FIG. 7A

FIG. 7B

Order data set

OrderNo	CustID	State	LineItems
9998	1000	CA	
9999	1001	CA	
9777	1202	TX	

Item	Qty	ItemPrice
001	2	10
002	4	5

Item	Qty	ItemPrice
001	7	23
002	7	10

Item	Qty	ItemPrice
001	9	99
002	1	2

Order data set with Group By State

State	LineItems
CA	
TX	

Item	Qty	ItemPrice
001	2	10
002	4	5
001	7	23
002	7	10

Item	Qty	ItemPrice
001	9	99
002	1	2

FIG. 8

Nested data set

OrderNo	CustID	LineItems
9999	1001	

Item	ItemQty
001	2
002	4

Header table

OrderNo	CustID	Item	ItemQty
9999	1001	001	2
9999	1001	002	4

FIG. 9A

Nested data set

OrderNo	CustID	LineItems
9999	1001	

Item	ItemQty
001	2
002	4

Header table

OrderNo	CustID
9999	1001

Line-item table

OrderNo	Item	ItemQty
9999	001	2
9999	002	4

FIG. 9B

Order data set

OrderNo	CustID	LineItems
9999	1001	

Item	ItemQty	ItemPrice
001	2	
002	4	

Qty	Sell Price
1	25
20	23

Qty	SellPrice
1	200
10	190

Unnested data set

OrderNo	CustID	Item	ItemQty	Qty	SellPrice
9999	1001	001	2	1	200
9999	1001	001	2	10	190
9999	1001	002	4	1	25
9999	1001	002	4	20	23

FIG. 9C

Nested data set

OrderNo	LineItems	CustInfo
9999		

Item	ItemQty
001	2
002	4

Type	CustID
Ship	1001
Bill	7777

Unnested data set

OrderNo	Item	ItemQty	Type	CustID
9999	001	2	Ship	1001
9999	002	4	Ship	1001
9999	001	2	Bill	7777
9999	002	4	Bill	7777

FIG. 10

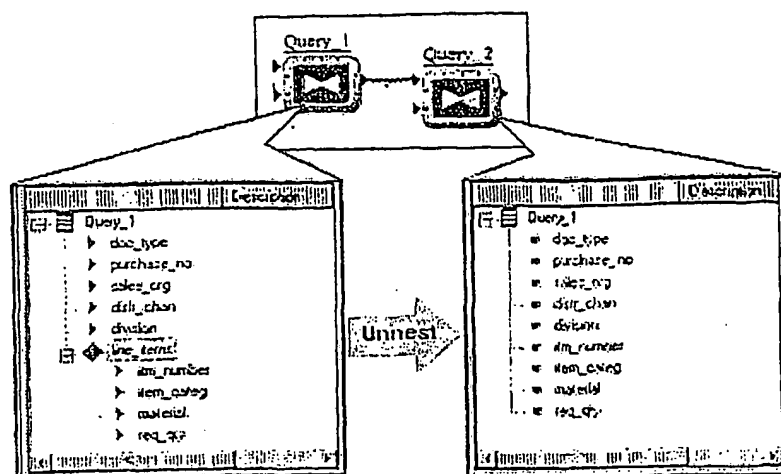


FIG. 11

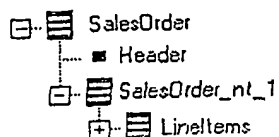


FIG. 12A

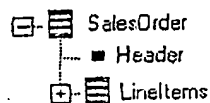


FIG. 12B

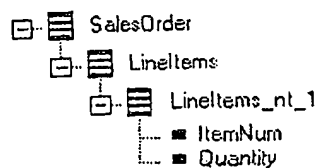


FIG. 12C

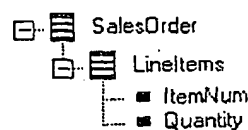


FIG. 12D

Message with data

OrderNo	CustID	ShipTo1	ShipTo2	LineItems
9999	1001	123 State St.	Town, CA	

Item	ItemQty	ItemPrice
001	2	10
002	4	5

Each column in the message corresponds to an ELEMENT definition in the DTD.

#### Corresponding DTD Definition

```
<?xml encoding='UTF-8'?>
<ELEMENT Order (OrderNo, CustID, ShipTo1, ShipTo2, LineItems+)>
<ELEMENT OrderNo (#PCDATA)>
<ELEMENT CustID (#PCDATA)>
<ELEMENT ShipTo1 (#PCDATA)>
<ELEMENT ShipTo2 (#PCDATA)>
<ELEMENT LineItems (Item, ItemQty, ItemPrice)>
<ELEMENT Item (#PCDATA)>
<ELEMENT ItemQty (#PCDATA)>
<ELEMENT ItemPrice (#PCDATA)>
```

FIG. 13

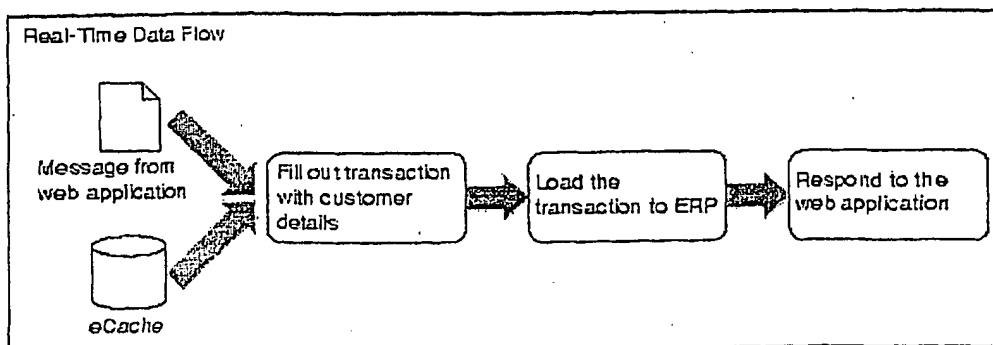


FIG. 14A

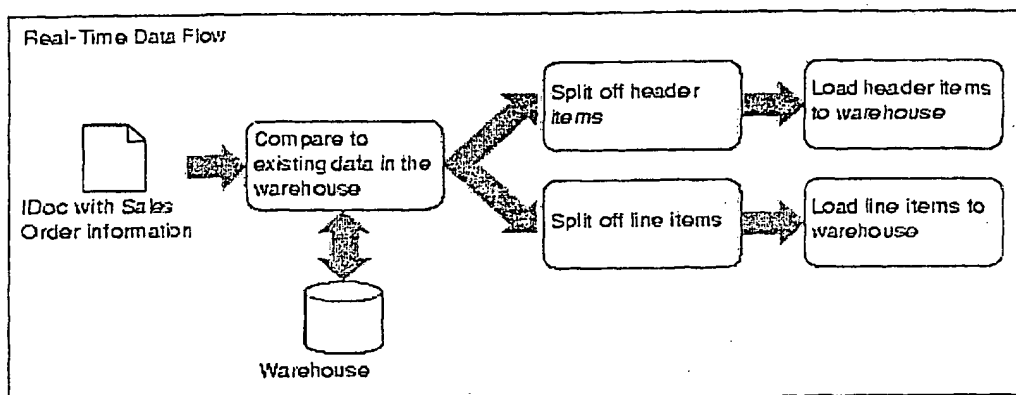


FIG. 14B

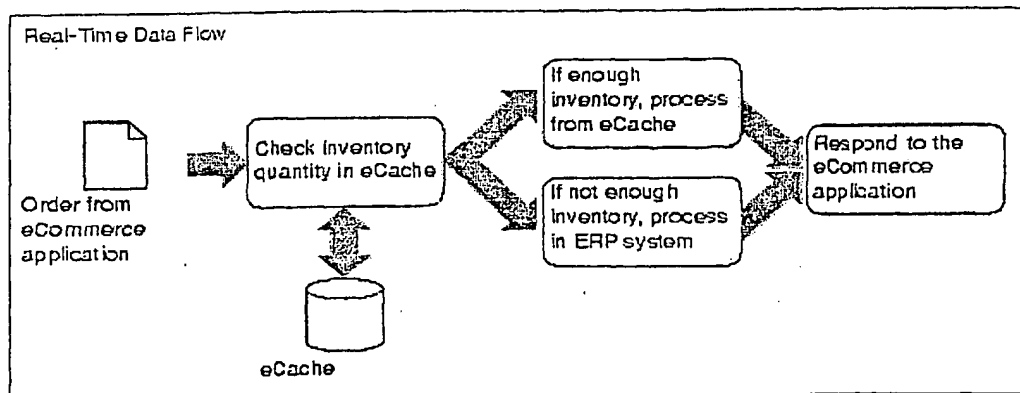
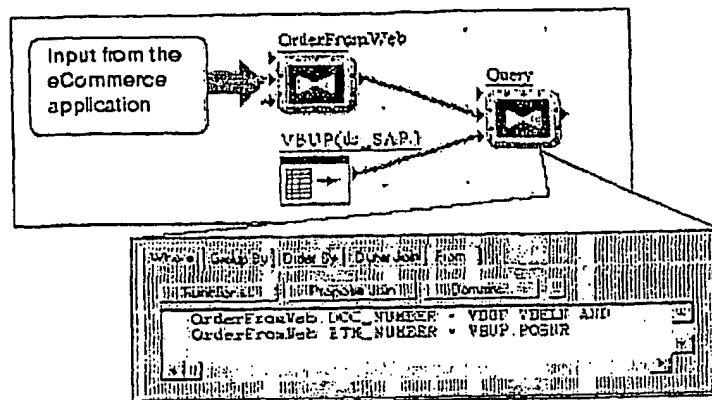


FIG. 14C

8/13



The WHERE clause joins the two inputs, resulting in output for only the sales document and line items included in the input from the eCommerce application.

FIG. 15

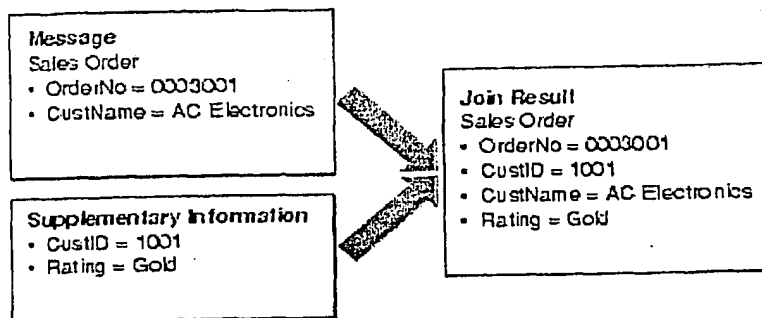


FIG. 16A

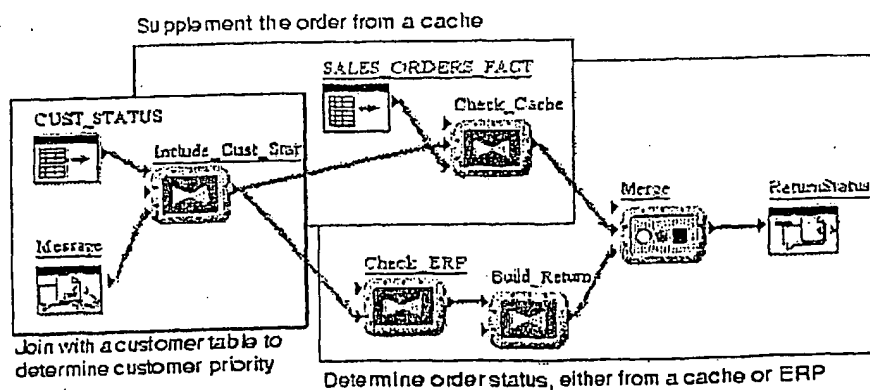


FIG. 16B

9/13

Incoming sales order

OrderID	CustID	LineItems
9999	1001	

Inventory data in cache

Material	Inv	IMargin
7333	500	100
2288	1500	200

Item	Material	Qty
001	7333	300
002	2288	1400

The quantity of items in the sales order is compared to inventory values in the cache.

FIG. 17

Incoming sales order

OrderID	CustID	LineItems
9999	1001	

Outgoing sales order

OrderID	CustID	LineItems
9999	1001	

Item	Material	Qty
001	7333	300
002	2288	1400

Item	Material	Qty	Inv
001	7333	300	
002	2288	1400	1300

FIG. 18

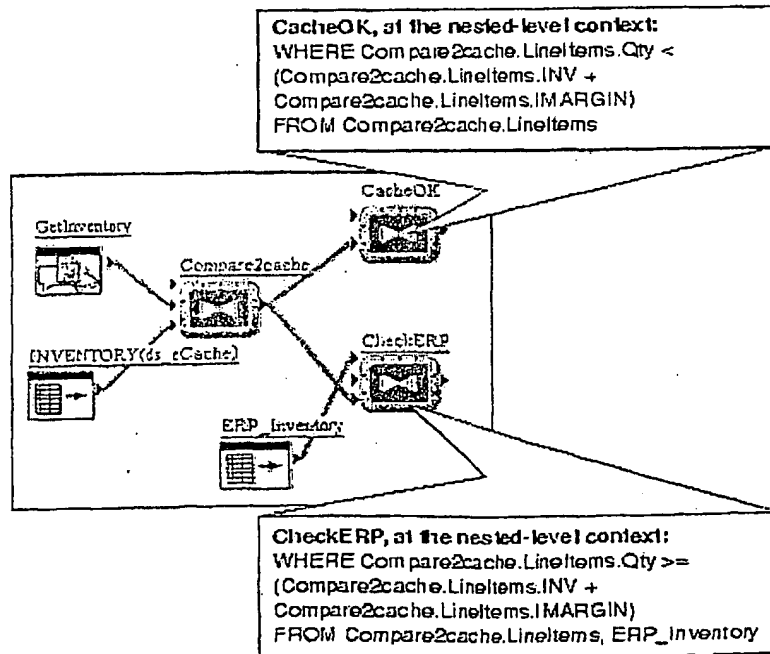


FIG. 19

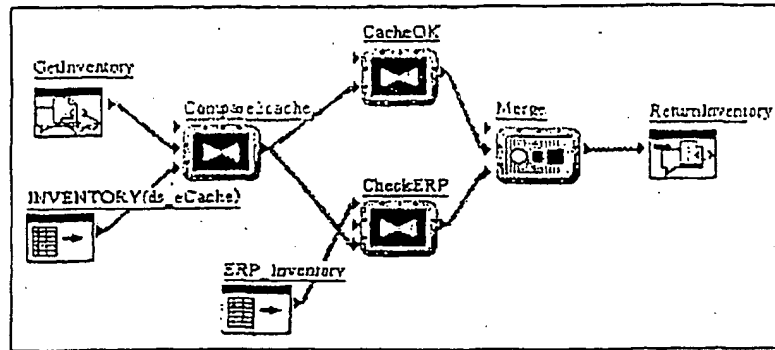


FIG. 20

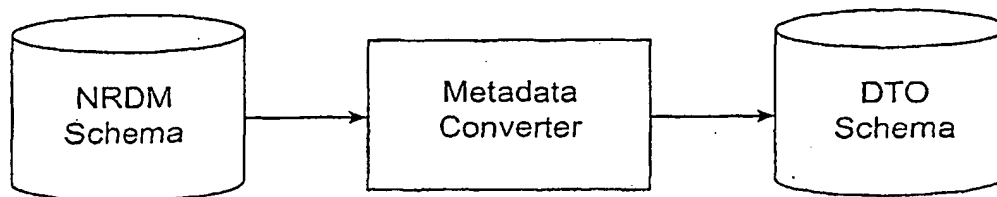


FIG. 21A

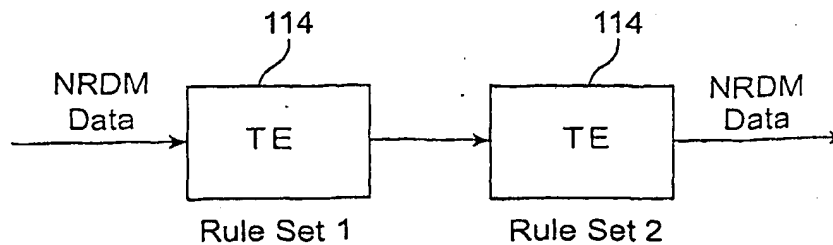


FIG. 21B

ORDER_ID	PROD_INFO				CID	CCITY
	PROD_ID	QTY	VENDOR_INFO			
			VNDR_ID	VNDR_CITY		
100	101	50	10SF	SanFran	444	SanFran
			20BK	Berkley		
	201	100	10SF	SanFran		
			30SJ	SanJose		
	301	100	30SC	SantaClara		
200	201	50	10SF	SanFran	555	Berkley
			30SJ	SanJose		
	301	100	30SC	SantaClara		
			20BK	Berkley		
300	401	50	35WC	WCreek	666	Dallas

FIG. 22

## VENDORS AND ORDERS TABLES

ORDER_ID	PROD_ID	QTY	CID	CCITY
100	101	50	444	SanFran
100	201	100	444	SanFran
100	301	100	444	SanFran
200	201	50	555	Berkley
200	301	100	555	Berkley
300	401	50	666	Dallas

PROD_ID	VNDR_ID	VNDR_CITY
101	10SF	SanFran
101	20BK	Berkley
201	10SF	SanFran
201	30SJ	SanJose
301	20BK	Berkley
301	10SF	SanFran
401	35WC	WCreek

FIG. 23

NUM_ORDERS	PROD_INFO	
	PROD_ID	QTY
3	101	50
	201	100
	301	100
	201	50
	301	100
	401	50

FIG. 24

ORDER_ID	PROD_INFO	
	PROD_ID	QTY
100	101	50
	201	100
	301	100
200	201	50
	301	100
300	401	50

FIG. 25

ORDER_ID	PROD_INFO	
	PROD_ID	QTY
100	201	100
	301	100
200	301	100
300		

FIG. 26

# INTERNATIONAL SEARCH REPORT

International application No.

PCT/US01/04698

## A. CLASSIFICATION OF SUBJECT MATTER

IPC(7) : G06F 17/00

US CL : 707/101

According to International Patent Classification (IPC) or to both national classification and IPC

## B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 707/100-104

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)  
EAST, ACM, IEL

## C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 5,937,409 A (WETHERBEE) 10 August 1999 (10.08.1999), ALL.	1-16
A	US 5,970,490 A (MORGENSTERN) 19 October 1999 (19.10.1999), ALL.	1-16

☐ Further documents are listed in the continuation of Box C.

☐ See patent family annex.

\* Special categories of cited documents:

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier application or patent published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T"

later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X"

document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y"

document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&"

document member of the same patent family

Date of the actual completion of the international search

27 March 2001 (27.03.2001)

Date of mailing of the international search report

Name and mailing address of the ISA/US

Commissioner of Patents and Trademarks  
Box PCT  
Washington, D.C. 20231

Facsimile No. (703)305-3230

Authorized officer

Thomas Black

Telephone No. 305-9000

*Peggy Hamed*

Form PCT/ISA/210 (second sheet) (July 1998)